DESIGN AND IMPLEMENTATION OF DS SPREAD SPECTRUM MULTIPLE ACCESS SYSTEM

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

SANJEEV KUMAR

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

June, 1992

CERTIFICATE

Certified that the work entitled, 'DESIGN AND IMPLEMENTATION OF DS SPREAD SPECTRUM MULTIPLE ACCESS SYSTEM' submitted by Mr. Sanjeev Kumar, Roll No. 9010440 has been carried out under my supervision and has not been submitted elsewhere for award of a degree.

June, 1992.

(V. Sinha)
Professor
Department of Electrical Engineering
Indian Institute of Technology
Kanpur - 208016

EE-1992-M-KUM-DES

114065

ACKNOWLEDGEMENTS

I am grateful to my thesis supervisor, Dr. V. Sinha, for suggesting the topic. His constant encouragement and constructive criticism helped a long way in shaping the thesis to the present form. I am thankful to Dr. P. K. Chatterjee for allowing me to work in Fibre Optics laboratory.

I thank Mahajan for his invaluable help at critical times. I acknowledge the help rendered by Karandikar. I am grateful to Manju, Mukesh and Ravi who inspired me to go for higher studies. I thank Bhatia, Galgali, Ballu, Tomar, Tyagi, Pankaj, Sujeet and other friends for their company during my stay at I.I.T. Kanpur.

Sanjeev Kumar

CONTENTS

ABSTRACT

Spread spectrum technique with its inherent interference attenuation capability, has ever the years become an increasingly popular technique for use in many different systems. In the present work, we have exploited the multiple access capability of the technique. A code division multiple-access (CDMA) system is not only used in military communications, but it also has varied commercial applications. Direct sequence technique has been chosen to achieve spreading and despreading the signal. Gold codes were used as direct sequences. Two separate transmitters for voice and data were developed. The receiver has been designed using Delay Lock Loop (DLL) technique. Hardware for transmitter was implemented and a software program for studying the performance of DLL receiver was developed.

LIST OF FIGURES

CHAPTER -1

INTRODUCTION

In present day communications, situations arise, when it becomes necessary for a communication system to resist external interference, to operate with a low energy spectral density, to provide multiple-access capability without external control, or to make it difficult for an unauthorized receiver to observe the message. In such a situation, it may be appropriate to sacrifice the efficiency (in terms of bandwidth utilization) aspects of the system in order to enhance these other features. Spread-spectrum technique offers a way to accomplish this objective.

The use of spread spectrum (SS) technique originally was confined to military communications and this application is still predominant. However, scenario is fast changing and a number of non-military applications are now increasing. Infact, there is growing interest in the use of this technique for mobile radio netowks (radio telephony, packet radio and amateaur radio) and timing and positioning system etc.

In a SS system information signal is spreaded over a wide frequency band, much wider than the minimum bandwidth required for transmission of information. On the receiver side, signal is despread and original information signal is recovered. A typical

spread spectrum (SS) digital communication system can be modelled as shown in Fig. 1.1.

In SS system, information signal is modulated with the help of a pseudo-random pattern and its bandwidth is expanded. On the receive side this very pseudo-random pattern is used to collapse the information signal into its original bandwidth. In other modulation scheme like FM, we expand the signal bandwidth but that is not spread spectrum technique. Because in spread spectrum technique spreading is achieved by a pseudo-random code, independent of information signal. This very nature of spread spectrum offers certain advantages, some of which are discussed in the next section.

1.2 APPLICATIONS OF SS TECHNIQUE

Some of the common applications of SS system are for

(i) Antijam capability

(ii) Interference rejection

(iii) Multiple access capability

(iv) Multipath protection

(v) Covert operation or low probability of intercept

(vi) Secure communication

(vii) Ranging.

Spread spectrum system resists the effects of intentional jamming. This antijam capability was, in fact, the primary reason for early consideration of spread spectrum communication by the military.

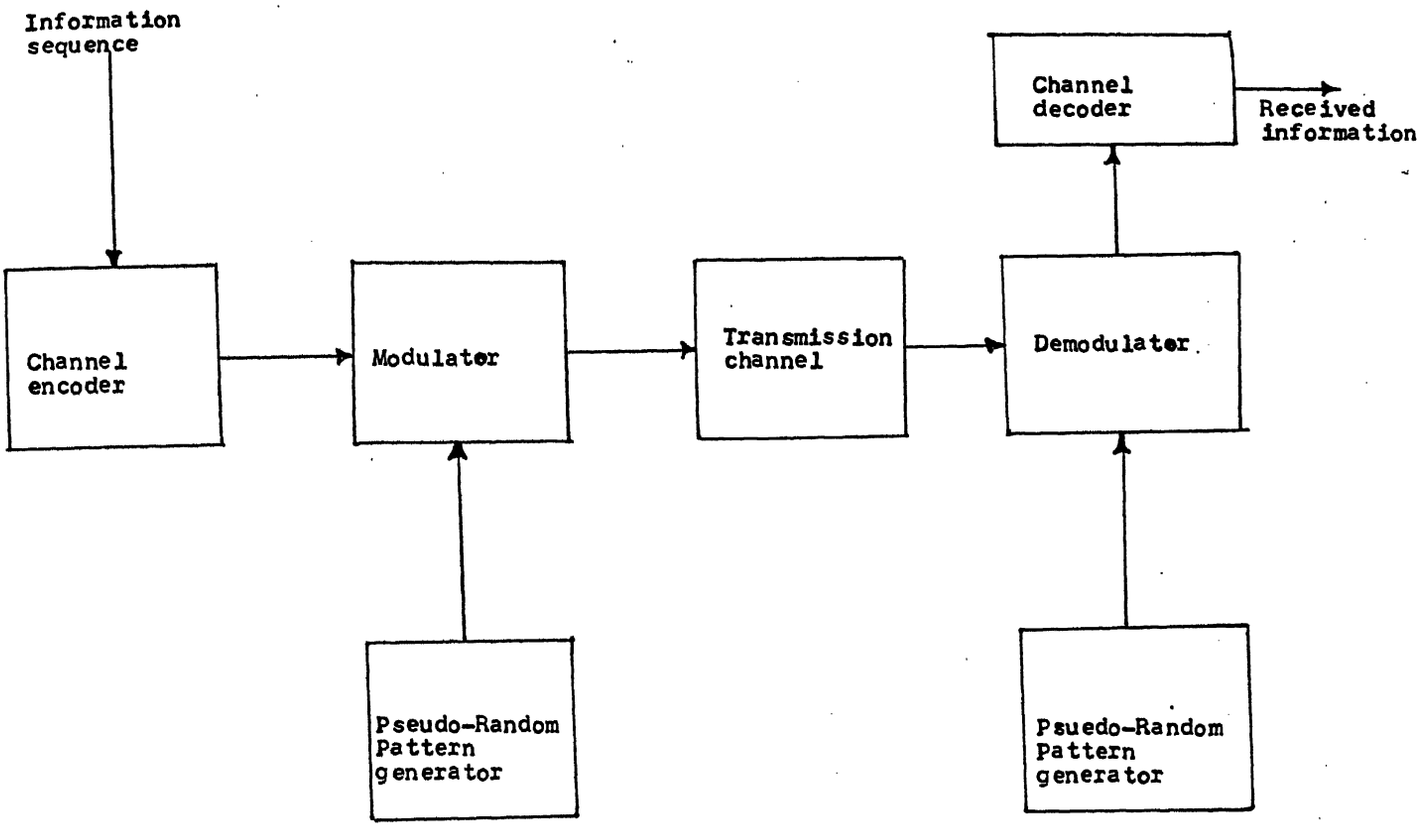


Fig. 1.1 A typical spread spectrum (SS) digital communication system.

An application of spread spectrum that is also of interest to the military is that of covert operation or low probability of intercept. In this case it is desired to maintain a signal spectral density that is sufficiently small so that its presence is not readily detected.

One of the most important application is usually referred to as multiple access. The need for multiple access arises when a number of independent users are required to convey their messages through a common facility. An example of this is satellite communication system in which all message from ground stations must pass through a common satellite repeator. Another such application is ground-based mobile communication in which mobile vehicle must communicate with a central base station. Another application of spread spectrum is usually referred to as selective calling. In a sense selective calling is the inverse of multiple access, that is, there is a central station that must communicate with a number of different receiving points and its wishes to do so on a selective basis. The message is for one receiving point and should not be received at other receiving points. An example of such a system might be ground-based radio system in which there is a central station that is communicating with a number of mobile receivers.

A characteristic of spread spectrum that is of particular interest in mobile communication is tha ability of a wide-band signal to resist the effect of multipath fading. A property of multipath fading is that frequencies separated by only a few

hundred kilohertz may fade essentially independently. Thus at any given time when the signal has a large bandwidth, only a small portion of the bandwidth will be in a fade. The average received signal power thus can be made more nearly constant than it would be for narrow-band signal. This resistance to fading is an important consideration in the potential application of spread spectrum to mobile communication situations.

Another advantage of using spread spectrum lies in its abilty to accomodate a secure communication system. If codes are made sufficiently complex, it becomes very difficult to break them, and for unauthorized listerns to determine what the message actually is.

A final application of SS lies in its ability to yield accurate distance information. It is well known that a broad-band signal can be resolved in time much more precisely than a narrow-band signal. Thus by transmitting a signal with a large bandwidth, it is possible to measure delay times much more accurately and obtain more accurate range information. This is of importance in radar and navigation systems.

A more comprehensive description of some of the above applications is given in Chapter 13 of [1].

1.3 TYPES OF SS SYSTEM

On the basis of modulation method spread spectrum system can be classified in following types :

(i) Direct sequence

(ii) Frequency hopping

(iii) Time hopping

(iv) Chirp

(v) Hybrid forms.

1.3.1 Direct Sequence

A typical direct-sequence transmitter is illustrated in Fig. 1.2. It contains a PN code generator that generate the pseudonoise sequence. The binary output of this code generator is added, modulo 2, to the binary message, and the sum is then used to modulate a carrier.

In the direct sequence receiver the received signal is despread correlating it which locally generated code sequence. It is important to achieve synchronization at the receiver. When synchronization is achieved, the output is simply the message. This is followed by the message demodulator. A typical DS, spread spectrum system receiver is shown in Fig. 1.3.

1.3.2 Frequency Hopping

In this method, transmitter is made to change from one frequency to another frequency in a pseudo-random manner, determined by a code sequence. The block diagram of a frequency-hopping transmitter is shown in Fig. 1.4. Frequency hopping is accomplished by means of a digital frequency synthesizer, which in turn is driven by a PN code generator.

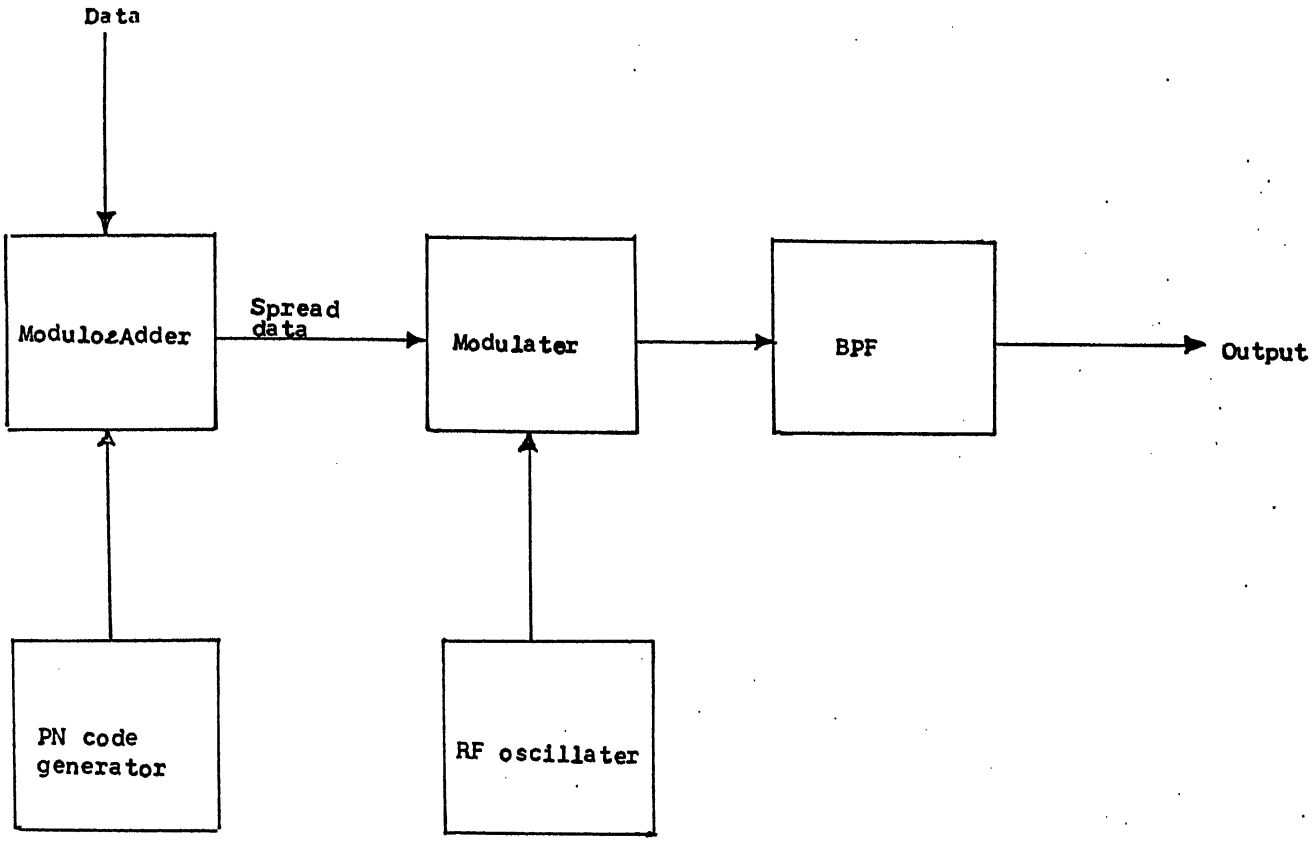


Fig. 1.2 DS spread spectrum transmitter.

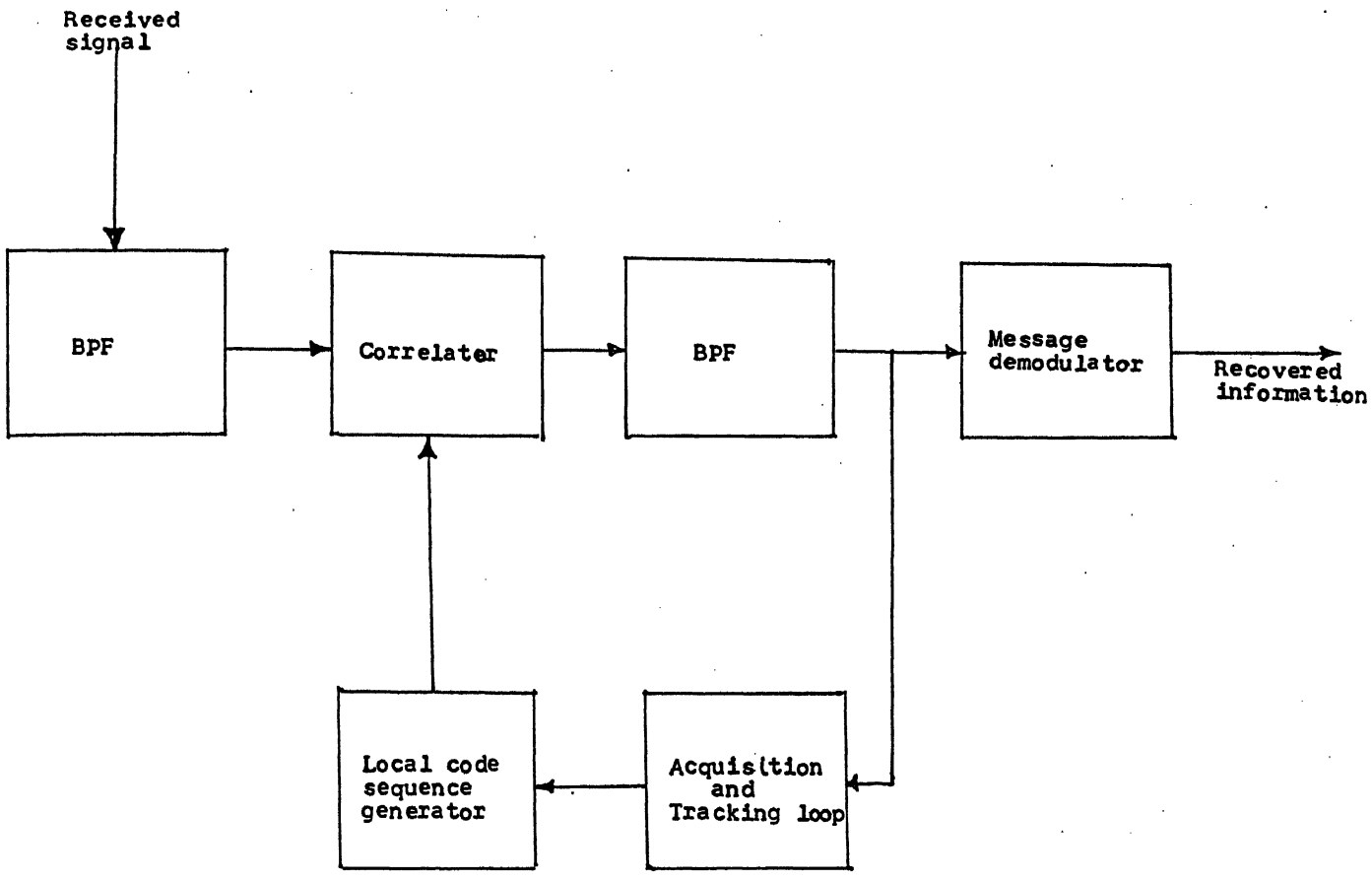


Fig. 1.3 DS spread spectrum receiver.

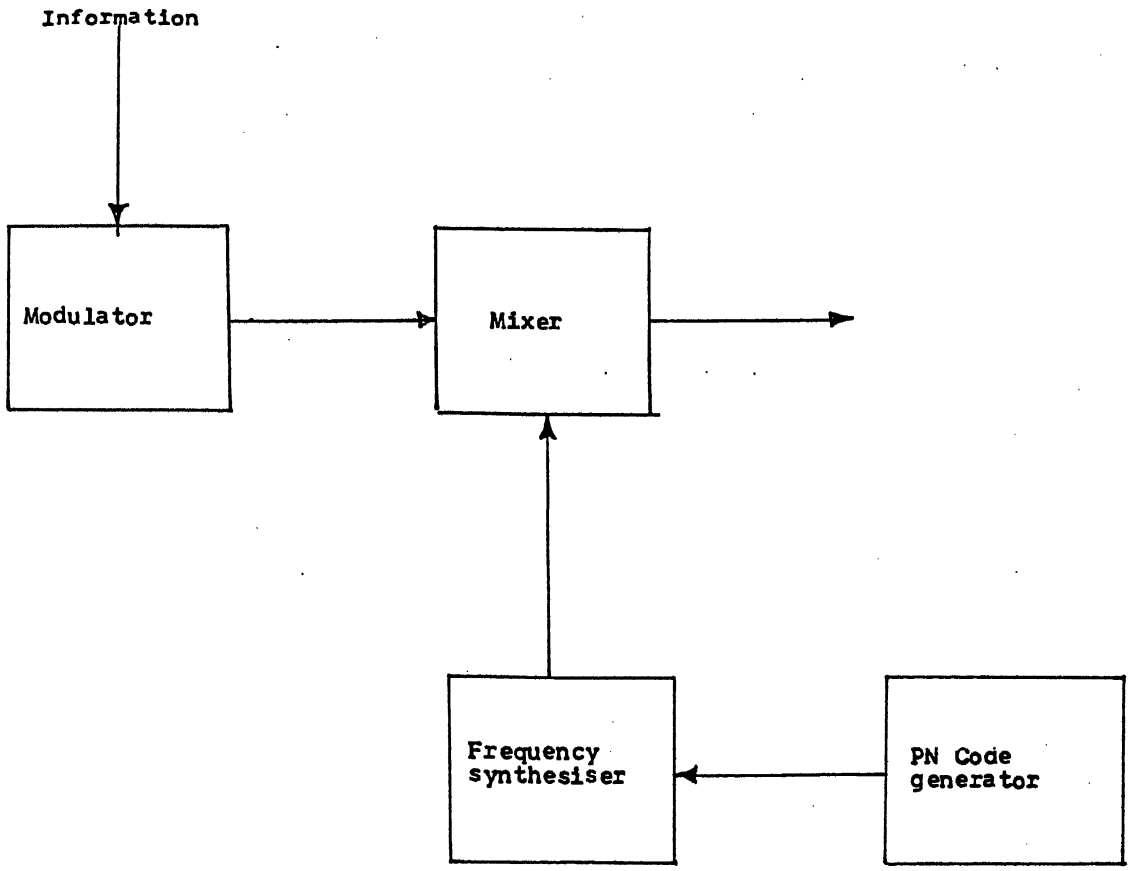


Fig. 1.4 FH spread spectrum transmitter.

In the FH receiver, a PN code identical to that one used at transmitter is used to generate a local reference signal. When the code generator controlling the frequency dehopping is in correct phase, the information is retrieved. Block diagram of a typical FH receiver is shown in Fig. 1.5. Code tracking is achieved by means of an early-late gate which is used to maintain a small timing error.

1.3.3 Time Hopping

Time hopping can be viewed as a form of spread spectrum which spreads the information in time slots and employs a burst transmission within the time slot. In this sense, it can be interpreated as dual of frequency hopping technique. This technique might be useful when a requirement is to operate the transmitter at high peak power but with a low duty factor. The selection of the slot in which the transmitted pulse occurs is controlled by a PN code generator. Block diagram of TH transmitter is shown in Fig. 1.6.

In the receiver the code tracking is performed on the data transitions by means of conventional bit synchronisers. Block diagram of TH receiver is shown in Fig. 1.7. Time hopping by its very nature is like time division multiple access technique and provides strong interference rejection from a nearby transmitter during its off time.

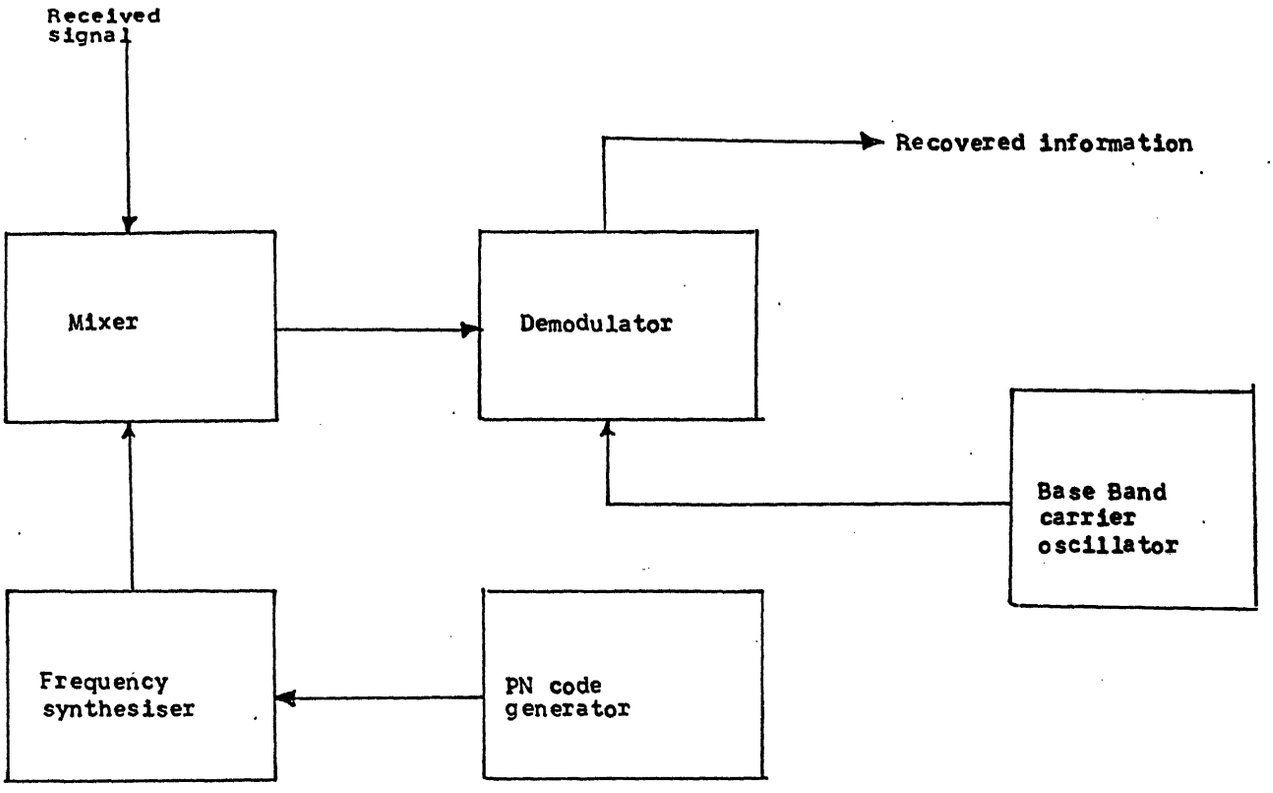


Fig. 1.5 FH spread spectrum receiver.

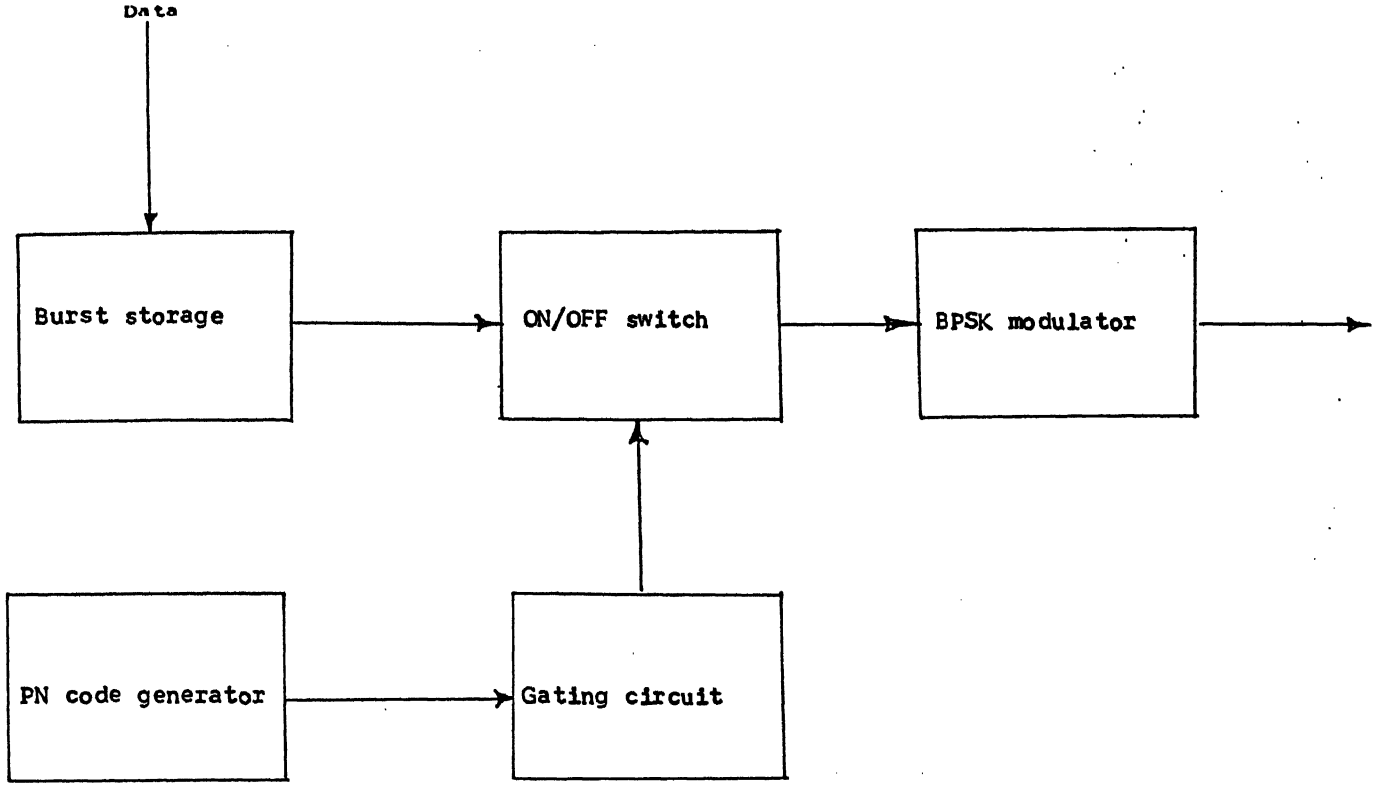


Fig. 1.6 TH spread spectrum transmitter.

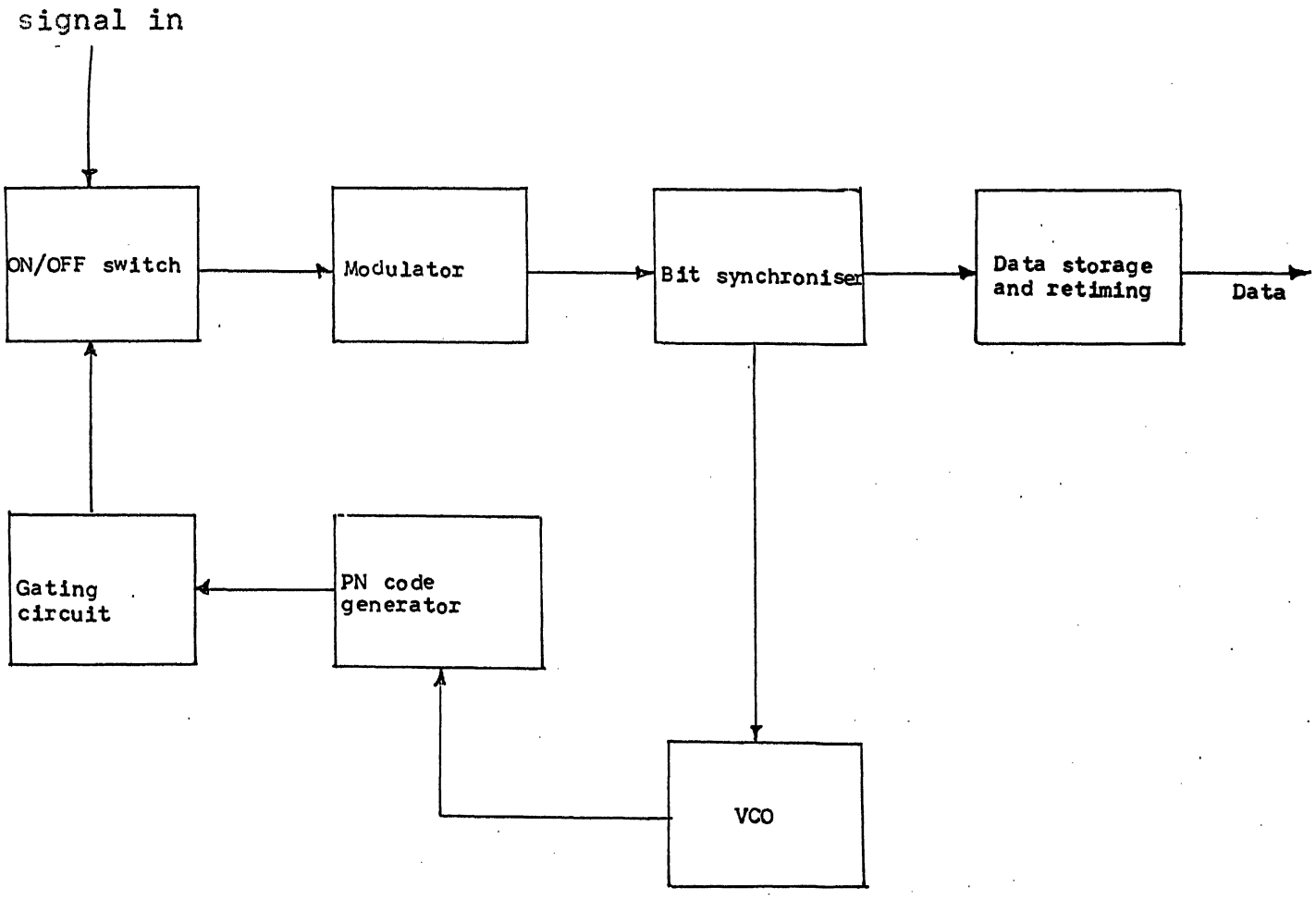


Fig. 1.7 TH spread spectrum receiver.

1.3.4 Chirp Spread Spectrum

A chirp spread-spectrum utilizes linear frequency modulation of the carrier to spread the bandwidth. This is a technique that is very common in radar systems. The relationship between frequency and time are shown in Fig. 1.8, in which T is the duration of given signal waveform and B is the bandwidth over which the frequency is varied. It is also possible to use nonlinear frequency modulation.

1.3.5 Hybrid Spread Spectrum System

The use of a hybrid system attempts to capitalize upon the advantage of a particular method while avoiding the disadvantage of it. Many hybrid combinations are possible, some of these are

DS|FH

DS|TH

FH|TH

DS|FH|TH

Implementation of hybrid forms are complex.

1.4 THESIS OUTLINE

SS system was first developed around the second world war. Originally SS technology was primarily meant for military applications and hence much of the literature was not readily available about research and development in SS technology. A comprehensive history of the development in SS techniques is presented by R.A. Scholtz [2].

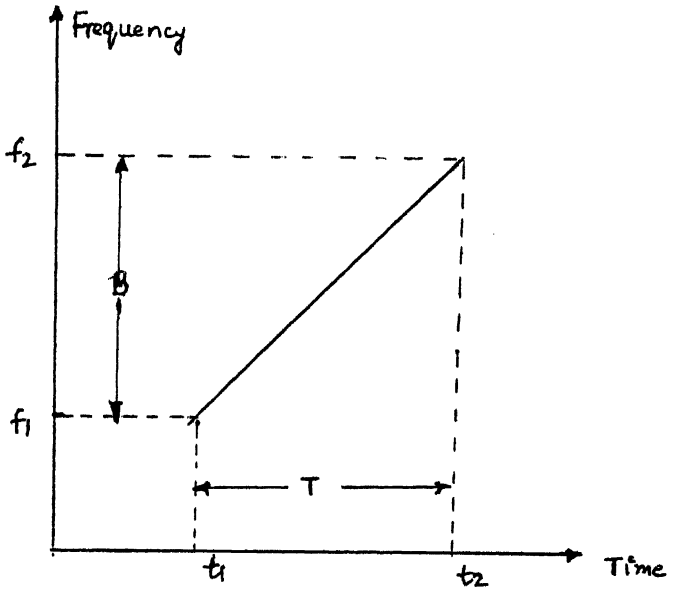


Fig. 1.8 Linear Chirp Signal.

Since the last decade SS techniques are getting attention for non-military applications also, and a lot of research effort to put this technology for commercial applications are underway. Research is aimed at commercial exploitation of SS techniques like mobile radio network, satellite applications and broadcasting operation. Appropriate pseudo-random codes are developed. Properties of PN code sequences are studied by P.V. Sarwate etc. [3]. For multiple access communication system, bound on number of user are developed [4], and performance criterion for multiple access communication system is developed [5]. Mobile radio network system are also developed. A such system is reported by Urs Grob et al. [6]. A system exploiting the broadcast capabilities of SS system is reported by E. Geraniotes et al. [7]. Development of integrated circuits (IC's) for SS systems are also attempted [8].

Inspite of recent research, practical implementation of SS system is not upto the level. Practical implementations is not much published in literature. In India, SS system is developed by SAC Ahmadabad. In our department also, a DS spread spectrum system was developed [9-10].

Aim of the present work is to exploit the capability of multiple access in SS system. For above purpose, a direct sequence (DS) spread spectrum system is reported in this thesis. Gold codes are used to spread the data. Information data of 64 kbps is modulated with a PN sequence of 4 Mbps. System is designed for less them seven users. NO RF modulation has been attempted.

Receiver used in present work is of delay locked loop (DLL) type. A software program is developed for studying the receiver structure.

The work is organized under following chapters.

In Chapter 2, fundamentals of DS spread spectrum system are discussed. Properties of PN codes are discussed and a receiver structure is presented. A specific multiple access application is discussed.

In Chapter 3, system development is reported. This chapter discusses the performance criterion of multiple access system. Structure for Gold codes are discussed next and a set of Gold codes of order 13 are developed. Receiver structure based on delay locked loop (DLL) is discussed towards the end of the chapter.

In Chapter 4, system realization is presented. In this chapter we have described the transmitter hardware. Software program for receiver is also described in this chapter.

And, finally in Chapter 5, concluding remarks regarding the system are presented. Suggestion for further development in the system are also contained in this chapter.

Appendices includes the circuit diagram of various sub-systems alongwith the software listing.

CHAPTER - 2

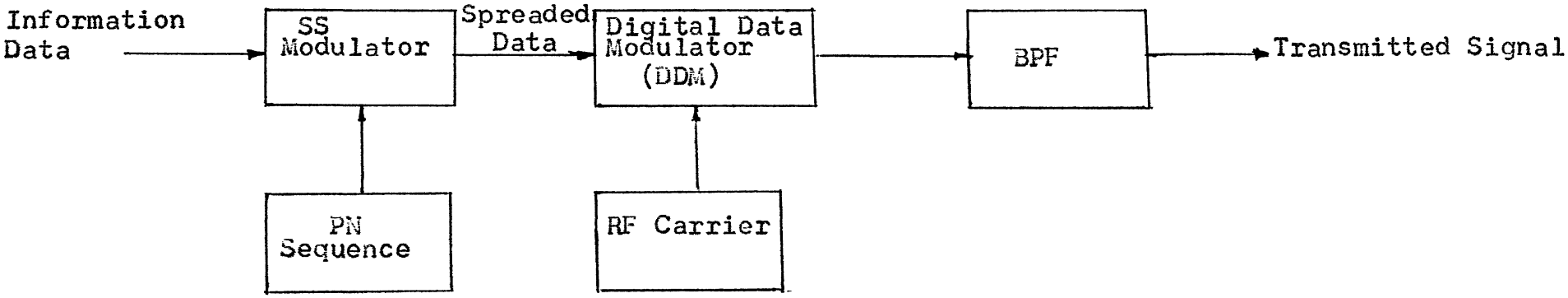
DS SPREAD SPECTRUM SYSTEM

Spread spectrum technique is a means of transmission in which the signal occupies a bandwidth in excess of the minimum necessary to send the information; the band spread is accomplished by means of a code which is independent of the data. At the receiver this very code is used synchronously for despreading the data. There are many advantages for spreading the information spectrum, for example, :- Antijamming, Antiinterference low probability of intercept, Multiple access, High resolution ranging etc., at the cost of increase in bandwidth and additional complexity. Some of the techniques for bandwidth spreading were discussed in the last chapter. One of them, namely, direct sequence method offers simplicity in design and implementation, hence, we will confine our discussion to direct sequence (DS) method for spreading the data.

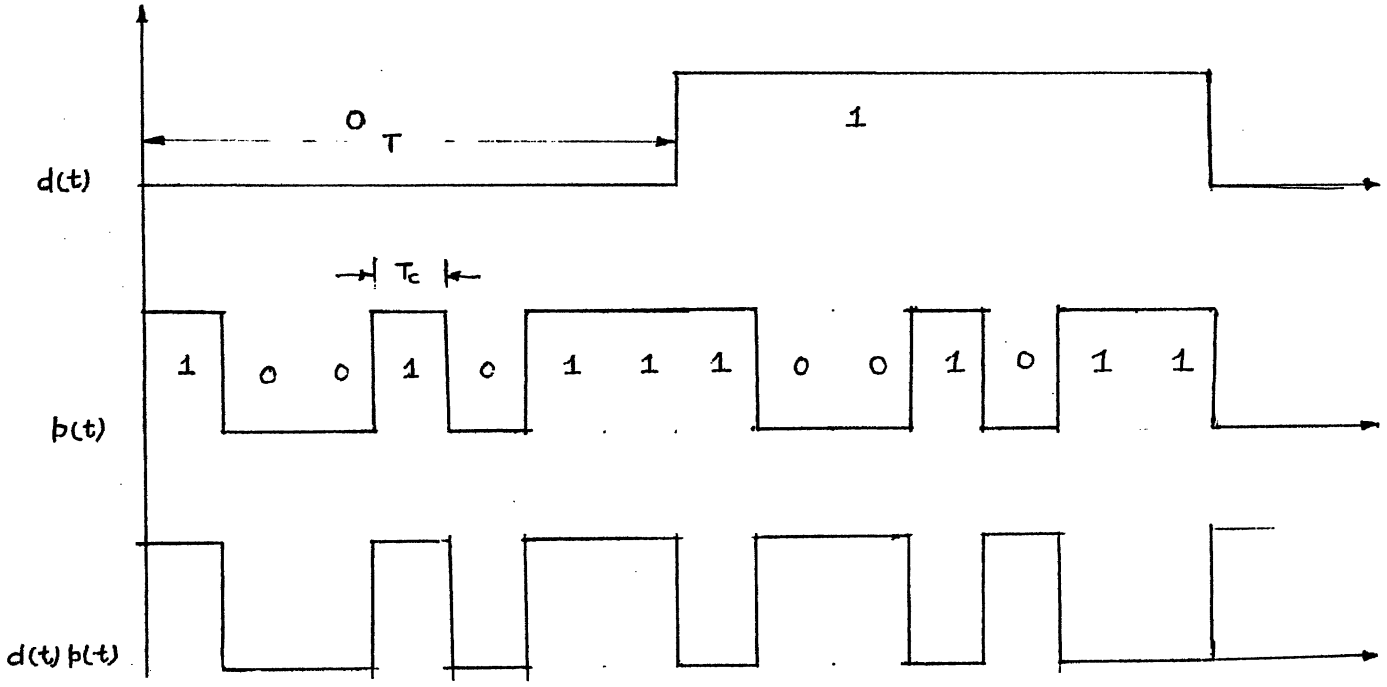
2.1 DS SPREAD SPECTRUM SYSTEM

Here we discuss direct sequence (DS) spread spectrum system. In the DS spread spectrum system the incoming data bits are modulated with a pseudo noise sequence. A simplified block schematic is shown in Fig. 2.1(a).

Here at the transmitter, information data bits are added to high speed code sequence and resultant spreaded data is used to



(a)



(b)

Fig. 2.1 (a) Block schematic of DS Spread Spectrum System.
(b) Timing diagram.

modulate an RF carrier. Modulator is a digital data modulator (DDM), for example BPSK, QPSK, MSK etc. Timing diagram of the system is shown in Fig. 2.1(b).

If information is represented by d(t) and spreading code by p(t) then the transmitted signal will be

$$S_t(t) = \sqrt{2}\, P\, d(t)\, p(t) \cos \omega_o t \qquad (2.1)$$

where f_o is the frequency of RF carrier and P is power of carrier.

Suppose the data is modulated by the r.f. carrier (i.e. no spreading) the single sided power spectral density of modulated data signal is given by

$$S_d(f) = \frac{1}{2} PT\, \mathrm{Sinc}^2 (f - f_o)\, T \qquad (2.2)$$

where T : bit duration of data.

The spectrum gets modified if the information is spread before modulation then the single sided power spectral density of spreaded signal is

$$S_t(f) = \frac{1}{2} PT_c\, \mathrm{Sinc}^2 (f - f_o)\, T_c \qquad (2.3)$$

where T_c : bit duration of code.

Both the power spectral densities ($S_d(f)$ and $S_t(f)$) are plotted in Fig. 2.2. It is evident that bandwidth of the information is increased by T/T_c. This is called the processing gain

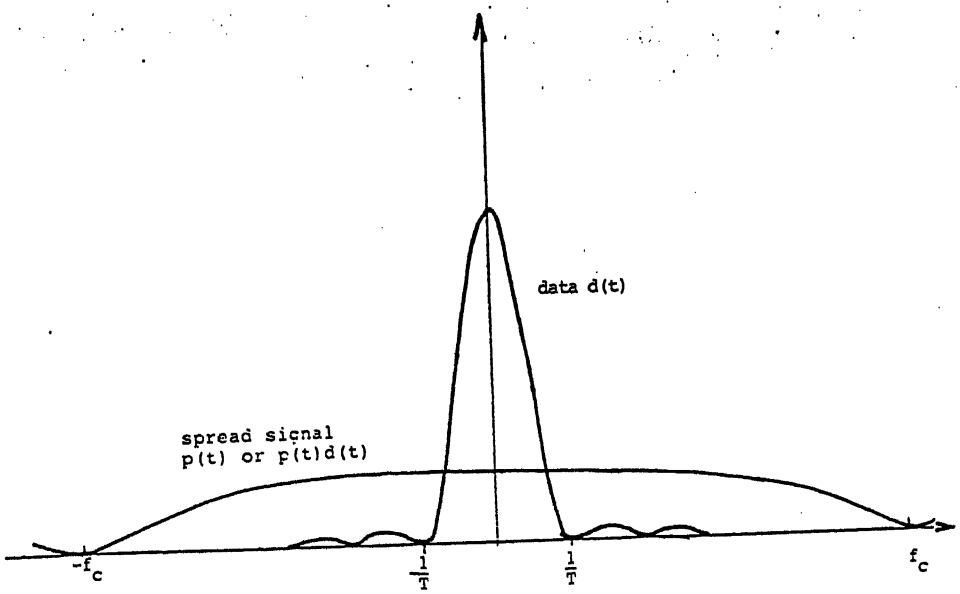


Fig. 2.2 Power spectral densities of data and of spreaded signal.

$$G_P = \frac{T}{T_c} = \frac{B_{SS}}{B_D} \qquad (2.4)$$

where B_{ss} = Bandwidth of the spreaded data

B_D = Bandwidth of the information data

The increase in bandwidth provides various advantages mentioned earlier, like protection against finite power interferring signal. The underlying principal is that of distributing a relatively low dimensional data signal in a high dimensional invironment [11]. The standard detection problem in communication involves transmission of M signalling waveforms. The transmitter select one of the waveform every T sec. to provide a data rate of $\log_2 M/T$ bit/sec. We say that the signal set s(t) (M signals) is D dimensional if the minimum number of orthonormal basis function required to define all the signal is D i.e.

$$S_i(t) = \sum_{k=1}^{D} S_{ik}\, \phi_k(t) \quad 0 \le t \le T \quad \text{for} \quad 1 \le i \le M \qquad (2.5)$$

where

$$S_{ik} = \int_0^T S_i(t)\, \phi_k(t)\, dt$$

and $\{\phi_k(t)\}$ $1 \le k \le D$ is an orthonormal basis set spanning the signal space such that

$$\int_0^T \phi_l(t)\, \phi_m(t)\, dt = \delta_{lm} \triangleq \begin{cases} 1 & l = m \\ 0 & l \ne m \end{cases} \qquad (2.6)$$

D is approximately equal to 2BT where B is total bandwidth (approximate) of the signal set. Spreading increases the dimensionality of the signal set (large 2BT). Therefore, if noise is a jammer with a fixed finite power, then spreading the signal bandwidth forces the jammer to distribute its finite power over larger number of dimensions, thereby reducing the effect of jamming [11].

Consider, received signal in a system

$$r(t) = d(t)\,p(t) + J(t) \quad 0 \le t \le T \tag{2.7}$$

d(t) : one data bit of duration T

p(t) : spreading code

J(t) : jamming signal with jamming power E_j

and

$$EJ = \int_0^T J^2(t)\,dt \tag{2.8}$$

If a receiver is a simple correlation receiver, the correlated receiver output would be

$$U(t) \overset{\Delta}{=} \sqrt{\frac{E_b}{T}} \int_0^T r(t)\,p(t)\,dt$$

$$= d(t) + \int_0^T J(t)\,p(t) \tag{2.9}$$

where E_b : bit energy.

Now, probability of error [11]

$$P_e \cong Q\left[\sqrt{\frac{E_b}{E_r}\, n}\right] \qquad (2.10)$$

$$Q(x) = \int_x^{\infty} \frac{1}{\sqrt{2\pi}}\, e^{-y^2/2}\, dy$$

where n is processing gain .

Therefore, processing gain can be seen as a multiplier of the "signal to jamming" ratio. Spreading the spectrum provides protection against broad-band jammer with a finite power P_j. Consider a system that transmitts R_o bit/sec. designed to operate over a bandwidth B_{SS} Hz over a AWGN channel with power density N_o W/Hz. For any bit rate R

$$\left[\frac{E_b}{No}\right] = \frac{P_S}{No_R} = \frac{P_S}{P_N}\,\frac{B_{SS}}{R} \qquad (2.11)$$

where $P_S \overset{\Delta}{=} E_b R$ = signal power

$P_N \overset{\Delta}{=} N_o B_{SS}$ = Noise power.

Then for rate R_o, minimum required $(E_b/N_o)_{min}$

$$\left[\frac{E_b}{No}\right]_{min} = \frac{P_S}{P_N}\,\frac{B_{SS}}{R_o} \qquad (2.12)$$

This is an expression for an ordinary case with no interferring signal. Now if a jammer with finite power P_j appaers, (we are

already transmitting at the maximum rate R_o) then the modified expression for E_b/N_o would be

$$\left(\frac{E_b}{No}\right)_{actual} = \frac{P_S}{P_N+P_j}\ \frac{B_{SS}}{R_o}$$

$$= \frac{P_S}{P_N+P_j}\ \frac{P_N}{P_S}\ \left(\frac{E_b}{N_o}\right)_{min}$$

$$= \frac{P_N}{P_N+P_j}\ \left(\frac{E_b}{N_o}\right)_{min}$$

$$\left(\frac{E_b}{N_o}\right)_{actual} = \left(\frac{E_b}{N_o}\right)_{min}\ \left(\frac{N_o}{N_o+P_j/B_{SS}}\right) \qquad (2.13)$$

Term

$$\frac{N_o}{N_o+P_j/B_{SS}} \rightarrow 1$$

as B_{SS} increases, thereby, retaining the performance, we had before jammer appeared. Fig. 2.3 shows the required value of processing gain needed to achieve the stated bit error probabilities for different P_J/P_C [12].

2.2 PN SEQUENCES

Ideally one needs purely random sequence to spread the data. But to despread the data we have to generate the replica of this sequence, thus purely random sequence becomes impractical. In practice, therefore, we use pseudo-random or pseudo-noise (PN) sequence so that the following properties are satisfied :

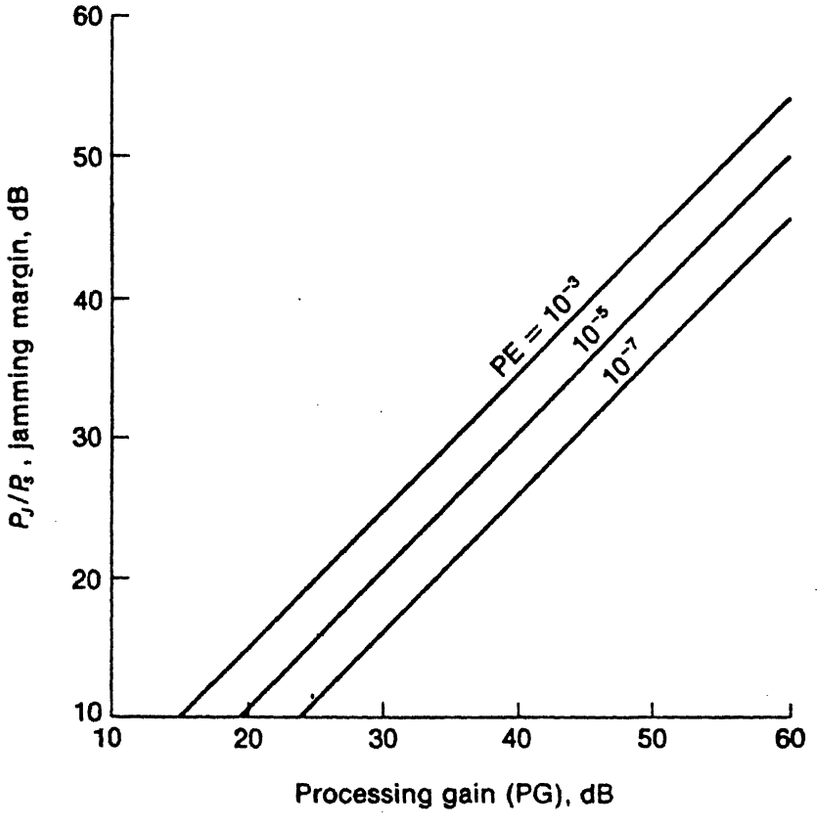


Fig. 2.3 Jamming performance.

(i) They have properties close to random ness

(ii) They have long period

(iii) They are difficult to construct from a short segment.

Maximal length linear feedback shift register (LFSR) sequences possesses property (ii) and most of properties (i) but not (iii). In addition, they are easy to generate. One canonical form of a binary LFSR is shown in Fig. 2.4.

Coefficient a_i are binary (0,1) and are chosen according to the primitive polynomial in GF(2). Recursive relation is given by

$$C_N = \sum_{k=1}^{N} a_k \; C_{N-k} \pmod 2, \quad a_N = 1$$

Basic equation of the register sequence is

$$C(x) = \frac{g(x)}{f(x)} \qquad \deg g(x) \leq N$$

f(x) is the characteristic polynomial of the register and this decides the connections in LFSR. g(x) depends on the initial condition of the register. C(x) is the generating polynomial and its coefficient is the generated sequence. For generating maximal length sequence, f(x) must be a primitive polynomial i.e. it should be irreducible polynomial in GF(2), and must divide $x^{2^N-1}+1$ then the period of ML sequence will be 2^N-1. Properties of ML sequence are

i) There is an approximate balance of zeros and ones (number of ones = number of zeros $\pm$ and 1).

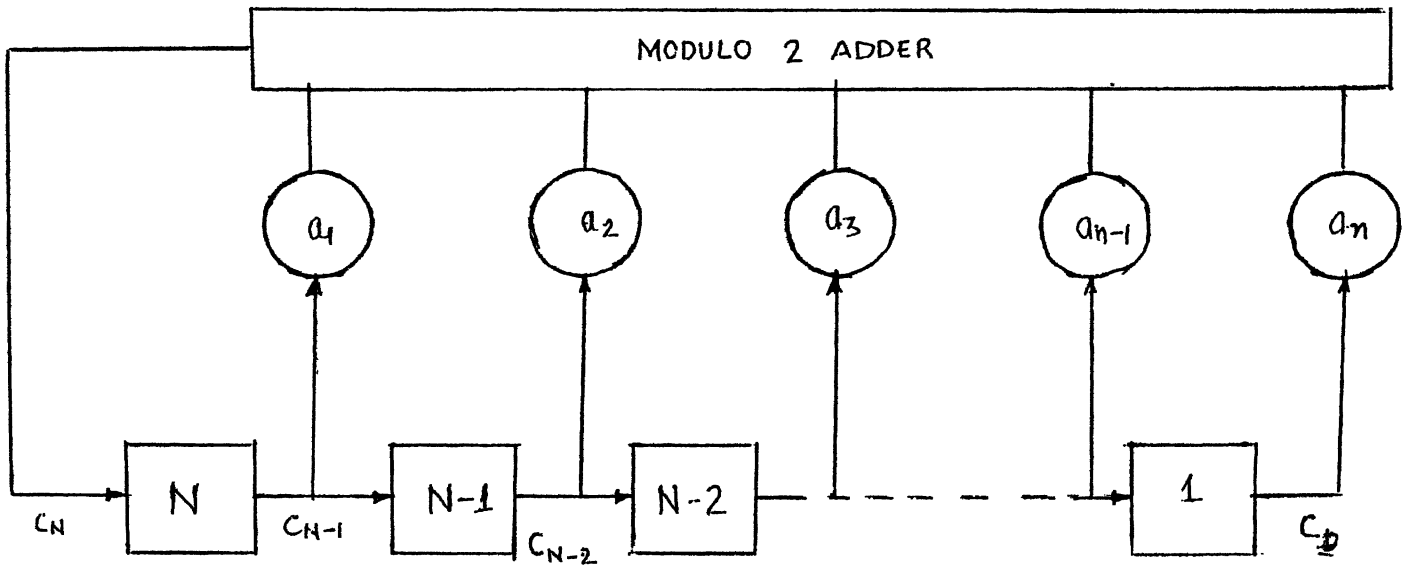


Fig. 2.4 Canonical form of a Binary LFSR.

ii) In any period, half of the runs of consecutive zero or ones, are of length one (1), one fourth are of length two etc.

iii)Auto-correlation function is

$$R_c(\tau) = \begin{cases} 1 & \text{for } \tau = 0, L, 2L... \\ -1/L & \text{otherwise} \end{cases}$$

where L = code period = 2^N-1.

Auto-correlation function is plotted in Fig. 2.5. Also shown is the frequency spectrum of ML sequences. If L is very large, the spetral lines get closer together, and for practical purposes, the spectrum may be viewed as being continuous [11].

2.3 RECEIVER STRUCTURE

In order to recover the information at receiver the receiver generates PN sequence in synchronizm with the received bit stream. The locally generated code has to be synchronized, both in frame and bit timing. The principle sub-blocks of a receiver are :

(i) Acquisition loop
(ii) Tracking circuit
(iii) Data recovery circuit

Fig. 2.6 shows the block schematic of synchronization system. Here the received signal is first locked into local PN signal generator using the acquisition circuit and then kept in

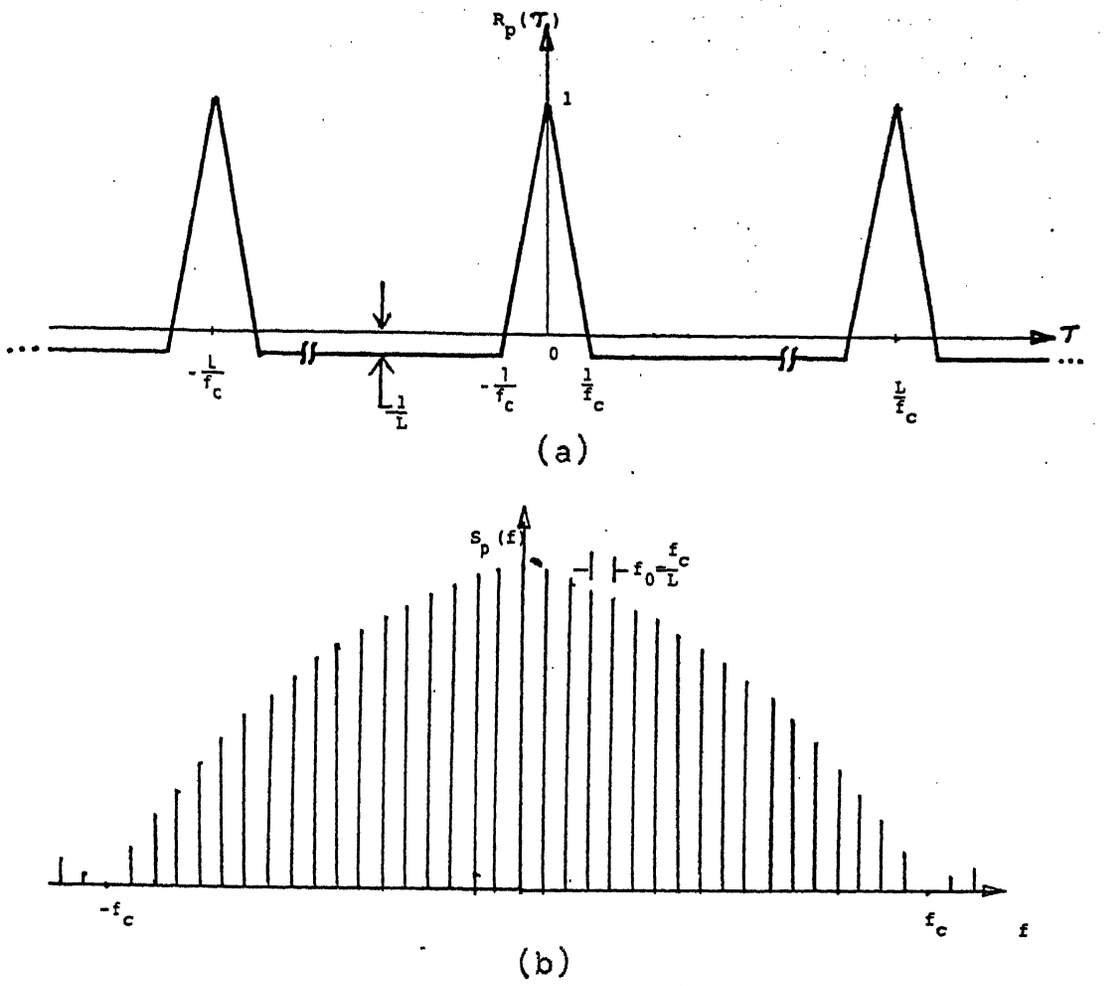


Fig. 2.5 Autocorrelation function and frequency spectrum of ML sequences.

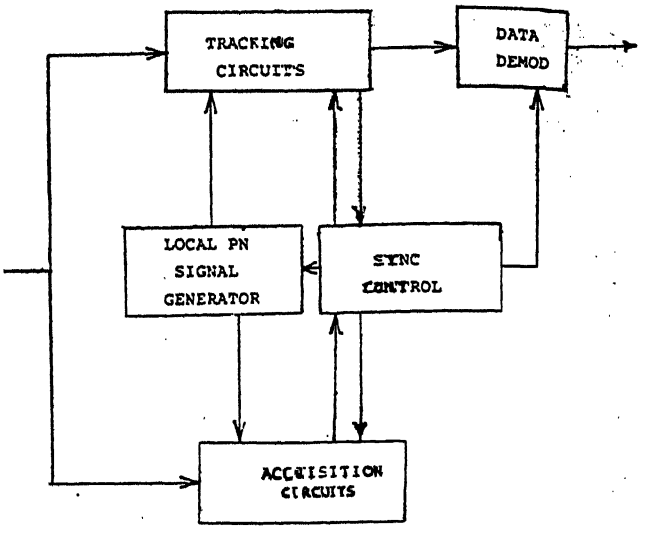


Fig. 2.6 Block schematic of synchronization system.

synchronism using the tracking system. Finally, data is demoulated

(i) One popular method of acquisition is the sliding correlator. In this system a single correlator is used. Initially, the output phase k of the local PN generator is set to k = 0 and a partial correlation is performed by examining λ chips (λT_c time). Then the correlated output is compared to some threshold value (V_T) and then if the correlator output falls below threshold, k is set to k = 1 and process is repeated with progressive value of k, till integrator output exceeds V_T. This technique is illustrated in Fig. 2.7.

(ii) Once acquisition, or coarse synchronization, has been accomplished tracking or fine synchronization takes place. One of the technique is shown in Fig. 2.8. Here, two separate correlators are used which are driven by two local code reference signals. They are identical but one is delayed version of the other. The amount of delay between the two local codes is usually one or two bits. Error signal is the difference of the outputs of the two correlators. Generated error signal drive the clock to align the locally generated code. Other method used in Tau-Dither tracking. It is a simple variation of DLL.

(iii) When locally generated code sequence is aligned with the received signal, the signal is multiplied (modulo 2 adder) with the tracked PN sequence. Resultant is passed to the digital data demodulator and massage is retrieved.

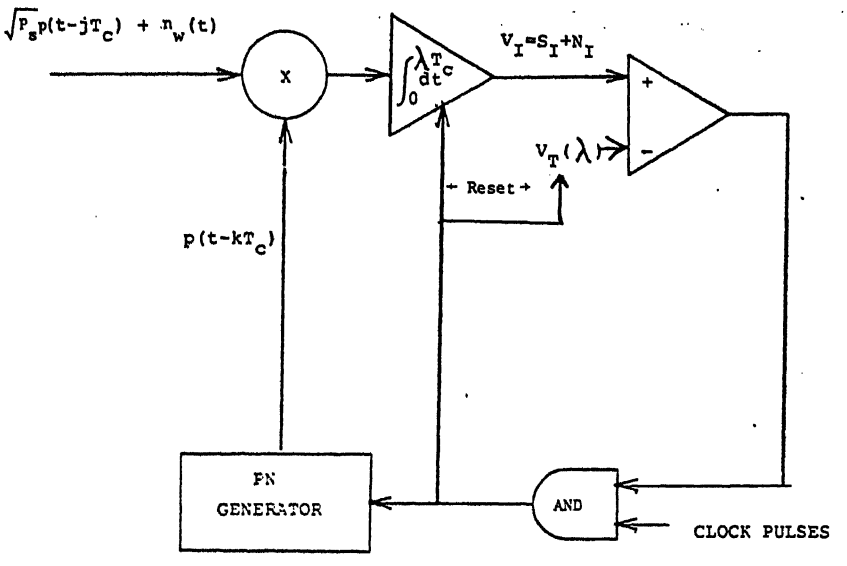


Fig. 2.7 The sliding correlator.

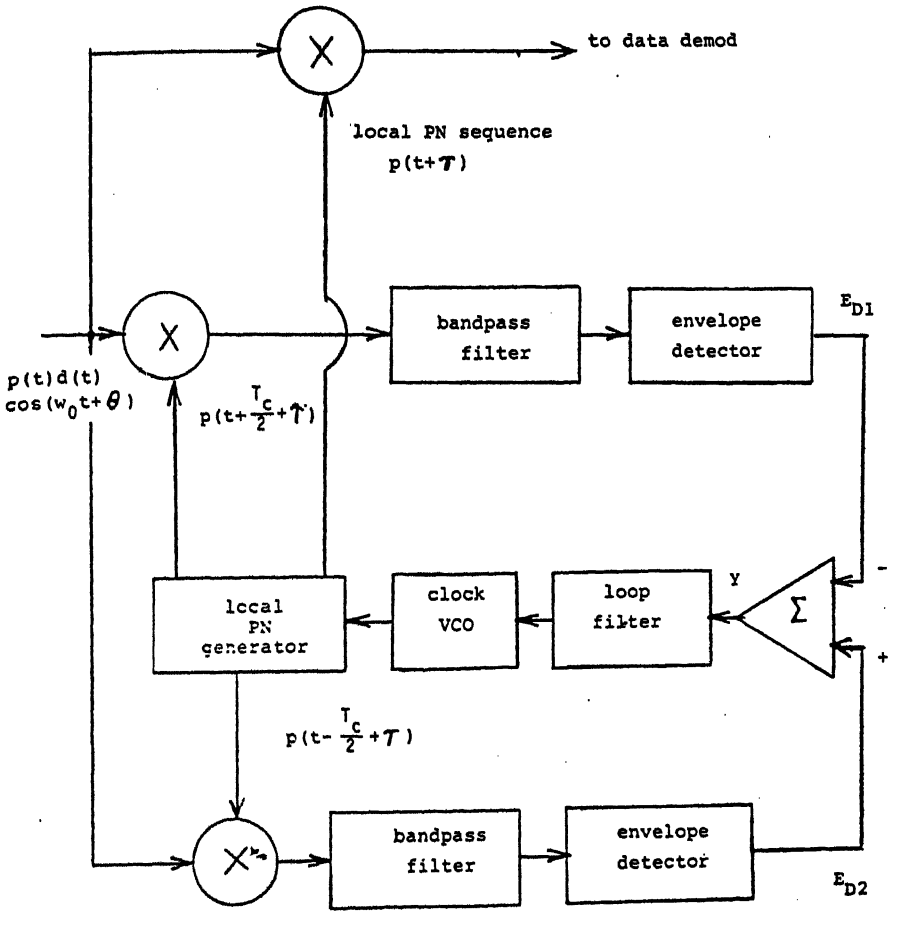


Fig. 2.8 Delay-locked loop for tracking direct sequence PN signals.

Acquisition time depends on the length of the correlator (λ chips). In turn λ depends on the desired probability of synchronization error. In the sliding correlator, for the worst case, we may have $k = 0,1,2,\ldots$ and 2^N-1. If during each correlation, λ chips are examined then worst case acquisition time will be

$$T_{acq,\ max} = 2\ \lambda \overset{L}{N} T_c$$

Mean acquisition time for sliding correlator is given by [11]

$$T_{avg} = \left[L(\lambda + \frac{1}{2})\ T_c + \frac{\lambda T_c P_F}{(1-P_F)^2}\right] + \frac{1-P_D}{P_D}\left[2L(\lambda + \frac{1}{2})T_c + \frac{\lambda T_c P_F}{(1-P_F)^2}\right]$$

P_D - probability of acquisition

P_F - probability of false alarm

L - Additional chip required after incorrect decision.

Acquisition time depends upon the intended application. Rapid acquisition is needed for present day communication applications.

Tracking circuit keeps the offset time τ small. If the error in tracking is τ then input to data demodulator is $p(t)\ p(t+\tau)\ d(t) \cos(\omega_o t + \theta)$. Demodulator removes the carrier and average the remaining signal. The result is $\overline{p(t)p(t+\tau)d(t)}$. Thus, amplitude of data has been reduced by $\overline{p(t)p(t+\tau)}$ and effective SNR at receiver decreases, therefore, prob. of error increases. Thus, error in tracking must be very small [11].

2.4 DS CODE DIVISION MULTIPLE ACCESS

The two most common multiple access techniques are frequency division multiple access (FDMA) and time division multiple access (TDMA). In FDMA, all users transmitt simultaneously but use disjoint frequency bands. In TDMA, all users occupy same RF bandwidth, but transmitt sequentially in time. In code division multiple access (CDMA) technique users transimitt simultaneously in time occupying same RF bandwidth.

In DS - CDMA, each user is given its own code which is approximately orthogonal (i.e. has low cross correlation) with the codes of other users. If codes can be found whose cross-correlations approach zero and whose auto-correlation are the energy of the signals, distinct multiple signals can be transmitted. The receiver can detect the desired signal while rejecting all other signals which do not match the code pattern. The key problem in DS CDMA is to find optimal codes.

A typical system configuration is shown in Fig. 2.9.

Received signal is

$$r(t) = \sum_{i=1}^{N} A_i \, d_i(t - \tau_i) \, p_i(t - \tau_i) \cos(\omega_o t + \theta_i) + n_w(t) \qquad (2.14)$$

where $d_i(t)$ = message of i th user

$p_i(t)$ = spreading sequence waveform of i th user

A_i = amplitude of i th user

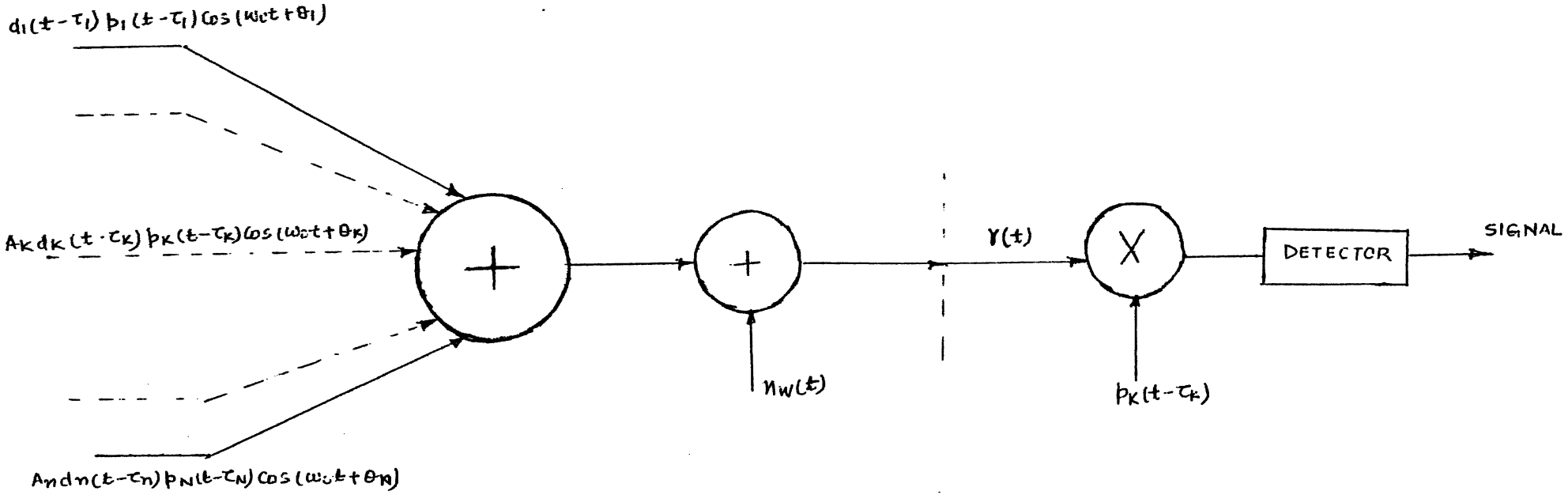


Fig. 2.9 DS CDMA System.

τ_i = random time delay of i th user waveform distributed uniformly in [0,T]

θ_i = random phase of i th carrier distributed uniformly in [0, 2π]

T = symbol duration

$n_w(t)$ = additive white Gaussian noise.

Assuming receiver is correctly synchronized to kth user than $\theta_k = 0$ and $\tau_k = 0$. Then correlator output at receiver

$$g(T) = \frac{1}{T} \int_0^T r(t)\, p_k(t)\, dt$$

$$= A_k + \frac{1}{T} \sum_{\substack{i=1 \\ i \neq k}}^{N} A_i \int_0^T d_i(t - \tau_i)\, p_i(t - \tau_i)\, p_k(t) \cos(\theta_i)\, dt$$

$$+ \frac{2}{T} \int_0^T n_w(t)\, p_k(t) \cos \omega_o t\, dt \qquad (2.15)$$

where double frequency terms has been ignored. Second term of equation (2.15) is the contribution due to the other users and this should be minimized. Therefore, the cross-correlation of codes should be minimum. Third term is due to while noise present in the channel. We shall look for methods which will minimize the effect of the second term.

In this chapter we discussed fundamental aspects of DS spread spectrum system. Then we discussed one specific application of SS in multiple access system. Basic structure for transmitter and receiver were discussed. With this much background, in next chapter we will develop the specific system suitable for multiple access application.

CHAPTER - 3

SYSTEM DEVELOPMENT

In the last chapter, we discussed some of the fundamentals of DS spread spectrum system. In this chapter we will develope the structure of various sub blocks for CDMA system. A simplified block diagram of CDMA system is shown in Fig. 3.1.

Here, two separate PN codes are used to spread voice and data. Both spreaded signals are transmitted over a common communication channel. On the receiver side, voice or data is demodulated depending upon the PN code used in receiver. Signals other than desired one behave as noise in the channel and, hence, degrade system performance. Thus, number of users becomes upper bounded. Constraint on number of users will be discussed in this chapter. Suitable PN codes will also be discussed.

3.1 PERFORMANCE CONSIDERATIONS OF CDMA

Number of users that can be accomodated in a DS CDMA system is a design parameter. In the last chapter, we discussed model for DS CDMA system and an expression for correlator output at receiver was obtained (eq. 2.15). Second term of this equation represents the signal from other users which can be considered as interferring noise so far as the signal for desired user is

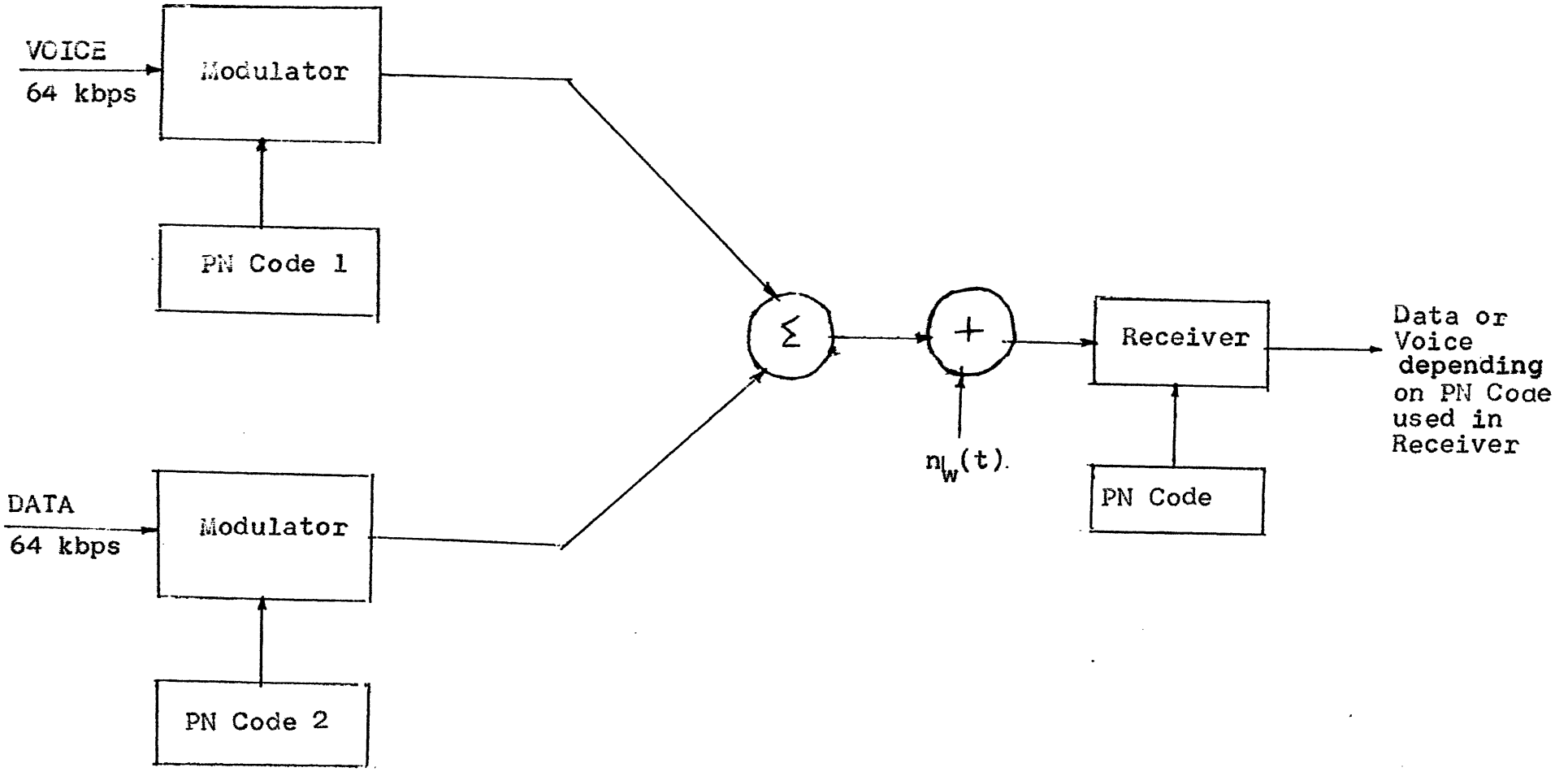


Fig. 3.1 Block Schematic of CDMA System.

concerned. As the number of users increases the interferring contribution due to this term increases which in turn adversely affect systems performance. Therefore, we can not increase the number of users beyond a certain limit. This limit is determined by the required system performance.

The effect of other users in the receiver matched to the desired signal is to be modelled. The chip rate is assumed to be much higher than the data rate. The code length is assumed to be sufficiently long so that at most one period of code occurs during each bit duration. When the code division sequences are purely random, the above assumptions allow us to conclude that effect of other users can effectively be modelled as gaussian random sequences.

When PN sequences are employed, situation is however somewhat more complex. During each bit period, the effect of each of the other user consists of a partial cross-correlation between the code sequence of the other users and that of desired signals. In fact, when BPSK data modulation is used, the effect of each of the other users may consist of the sum of two such partial period cross-correlations [5]. Partial cross correlation of the sequence p_i and p_k over a period τ_i is defined as

$$R_{ik}(\tau_i) \overset{\Delta}{=} \frac{1}{\tau} \int_{o}^{T_i} p_i(t - \tau_i)\, p_k(t)\, dt$$

Weber et al. [5] give the upper and lower bound on the probability function for PN sequences and Gold sequences. They conclude that for the set of Gold sequences of degree 13, effect of the sum of other users in the receiver matched to the desired signal can effectively be modelled as a Gaussian random variable. We use Gold sequences of degree 13 in our work. Based on these assumptions, signals due to other users are nothing but additive white Gaussian noise so far as the desired signal is concerned. The bit energy to noise density ratio is obtained as [5]

$$\left[E_b/N_o\right]_n = \frac{(E_b/N_o)_1}{\left\{ 1 + \frac{1}{N_o} \sum_{i=1}^{n-1} \alpha_i P_o T_c \right\}} \qquad (3.1)$$

where $(E_b/N_o)_n$ represents the actual signal energy per data bit to the one sided noise power spectral density (PSD) for the receiver matched to desired signal when n users are present $(E_b/N_o)_1$ represents the bit energy to single sided noise density that would have been obtained for a single user

P_o is power of the desired user

P_i is power of the i th user

$\alpha_i = P_i/P_o$

τ_i = chip time or pseudo random code bit period.

We want the performance of the n user system to be same as performance of single user. For above purpose we have to increase

$(E_b/N_o)_n$ to $(E_b/N_o)_1$. Let the required value of (E_b/N_o) single user is $(E_b/N_o)_R$ then

$$\left[E_b/N_o\right]_n = \frac{(E_b/N_o)_R}{\left\{1 + G^{-1}(E_b/N_o)_R \sum_{i=1}^{n-1} \alpha_i\right.}$$

$$= \left\{\left[E_b/N_o\right]_R^{-1} + G^{-1}\left[\sum_{i=1}^{n-1} \alpha_i\right]\right\}^{-1} \qquad (3.2)$$

where G = processing gain = $R_c/R_s = T_s/T_c$.

Now setting $(E_b/N_o)_n = (E_b/N_o)_1$, the required SNR can be solved

$$\left[E_b/N_o\right]_R = \frac{(E_b/N_o)_1}{1 - G^{-1}(E_b/N_o)_1 \sum_{i=1}^{n-1} \alpha_i} \qquad (3.3)$$

So, increase in (E_b/N_o) to maintain same probability of error can be considered as degradation factor (DF) and is defined as

$$DF = \frac{(E_b/N_o)_R}{(E_b/N_o)_1} = \frac{1}{1 - G^{-1}(E_b/N_o)_1 \sum_{i=1}^{n-1} \alpha_i} \qquad (3.4)$$

It can be observed from eqn. (3.2) that as number of users n increases $(E_b/N_o)_R^{-1}$ must decreases in order to keep $(E_b/N_o)_n$,

hence probability of error, constant. But ultimately, however, no amount of increase in $(E_b/N_o)_R$ can offset the increase in term

$$G^{-1} \sum_{i=1}^{n=1} \alpha_i$$

This results in limitation on number of users.

Now, let all the n users have same power so $\alpha_i = 1$ for all i

from 3.4 $$DF = \frac{1}{1 - G^{-1} (n-1) (E_b/N_o)_1}$$

from 3.2 $$(E_b/N_o)_n = \frac{(E_b/N_o)_R}{1 + G^{-1} (E_b/N_o)_R (n-1)}$$

for large values of $(E_b/N_o)_R$

$$\lim_{(E_b/N_o)_R \to \infty} (E_b/N_o)_n = \frac{G}{n-1} \quad ; \quad n \geq 2$$

Define, multiple access capacity factor (MACF) as

$$MACF \overset{\Delta}{=} \frac{G}{(N-1)} \left[E_b/N_o\right]_R^{-1}$$

If MACF > 1 then $(E_b/N_o)_n$ can be achieved by increasing $(E_b/N_o)_R$ but if MACF < 1 then no value of $(E_b/N_o)_R$ will give desired value of $(E_b/N_o)_n$. So number of users are limited by condition MACF > 1.

In this work G = 64 (f_c = 4 MHz, Data rate = 64 kbps) for probability of error $P_b = 10^{-6}$,

$$\left[E_b/N_o\right]_n = 10.5 \text{ dB} = 11.22$$

$$\text{MACF} = \frac{64}{n-1} (11.22)^{-1}$$

$$\Rightarrow \quad n - 1 \leq 5.7$$

$$\Rightarrow \quad n \leq 6.7 \quad \Rightarrow \quad n < 7$$

Thus, the number of users are upper bounded by 7. For increasing n we can increases the processing gain G. A plot of MACF and DF with number of users is shown in Fig. 3.2. Curve shows the two cases for $P_b = 10^{-5}$ and 10^{-6} clearly, we can trade off the P_b with the number of users. It is also evident that higher value of processing gain (G) can accomodate larger number of users.

3.2 GOLD CODES

One of the design objective in demodulation is to minimize the effect of interference due to other users. For this, we require codes which have uniformly low cross-correlation values i.e. for any two codes P', P" of code set

$$R_{P',P"}(\tau) \overset{\Delta}{=} \sum_{k=1}^{L} P'_k \; P"_{kT_c}$$

should be << 1 for all values of τ. Here L is code length.

Gold codes are such codes, hence, are suitable for CDMA application. The underlying principal of these codes is based on the following theorems (12, Chapter 11, Page 550-552).

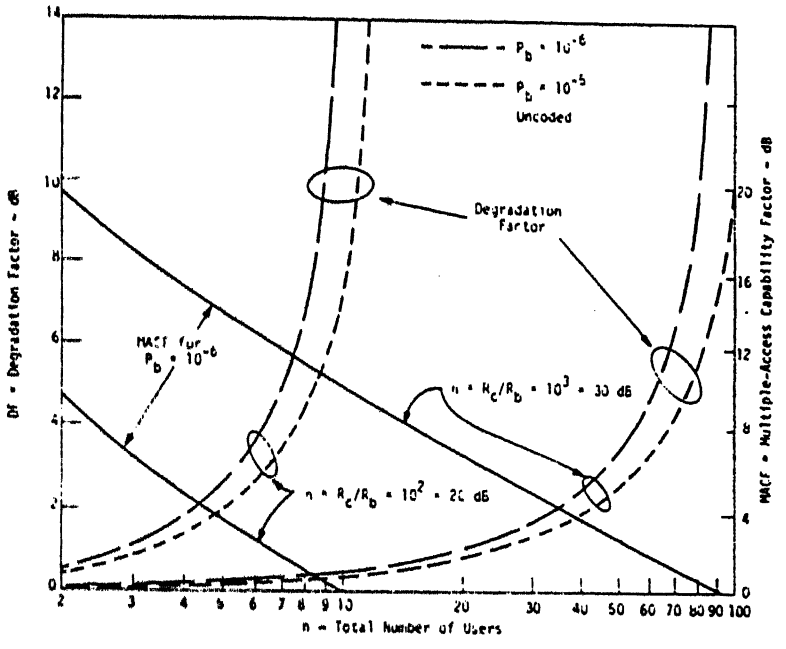


Fig. 3.2 System Performance for n equal power users.

Theorem 1

Let f(x) and g(x) be a preferred pair of primitive polynomials of degree n whose corresponding shift register generate maximal length sequence of period 2^n-1 and whose cross-correlation function R(k) satisfies the inequality

$$|R(k)| \leq \begin{cases} 2^{\frac{(n+1)}{2}} + 1 & \text{for n odd} \\ 2^{\frac{n+2}{2}} + 1 & \text{for n even,} \quad n \neq 0 \bmod 4 \end{cases}$$

then the shift register corresponding to the product polynomial f(x) g(x) will generate 2^n+1 difference sequence, each of period 2^n-1 and such that the cross-correlation of any pair satisfies the above inequality.

All 2^n+1 distinct codes are called Gold codes and are of period 2^n-1.

Above mentioned preferred pair of primitive polynomial can be generated with the help of following theorem.

Theorem 2

Let f(x) be a primitive polynomal of degree n such that n is not divisible by 4. Let α be a root of f(x), that is, $f(\alpha) = 0$. Let g(x) be the irreducible polynomial such that

$\alpha^{(2^{(n-1)/2} + 1)}$ is a root of g(x) for n odd

$\alpha^{(2^{(n-2)/2} + 1)}$ is a root of g(x) for n even.

then cross-correlation function of the sequence generated by f(x) and g(x) satifies following inequalities :

$$|R(k)| \leq 2^{(n+1)/2} + 1 \qquad \text{for n odd}$$

$$|R(k)| \leq 2^{(n+2)/2} + 1 \qquad \text{for n even and } n \neq 0 \bmod 4.$$

3.2.1 Balance Gold Codes

A balance code is one in which the number of "ones" exceeds the number of "zeros" by one only. In DS CDMA we need balanced codes but not all of the above discussed Gold codes are balanced. Number of balance Gold code is $2^{n-1}+1$. Balanced Gold codes are generated by selecting the proper relative phases of the two original maximal sequences.

Initial conditions for balanced Gold codes are [12].

If f(x) be desired polynomial then define h(x)

$$h(x) = \frac{d(x\ f(x)}{dx} \qquad n \text{ odd}$$

$$= f(x) + \frac{d(x\ f(x)}{dx} \qquad n \neq 0 \bmod 4$$

then initial conditions are obtained by long division of h(x)/f(x) to provide the first n coefficients, which are, in fact, the n initial conditions.

The initial conditions for the shift register corresponding to g(x) are subject to the constraint that the first stage contains a zero.

3.2.2 Realization of Gold Code

Configuration of Gold code generator is shown in Fig. 3.3. By changing the initial conditions of g(x) shift regsiter (subject to constraint of zero in first stage). We can generate number of balanced Gold codes (infact $2^{n-1}+1$ codes) which can be used for different users.

In this work, n = 13, number of balanced Gold codes is $2^{12}+1 = 4097$. Period of code sequences is $= 2^{13}-1 = 8191$. Let the primitive polynomial of order 13 be [13]

$$f(x) = 1 + x + x^3 + x^4 + x^{13}$$

Since n is odd, root of preferred pair polynomial g(x)

$$\alpha^{2^{(13-1)/2}+1} = \alpha^{65}$$

From the table of polynomial, we get

$$g(x) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{10} + x^{12} + x^{13}$$

Initial conditions for register 2 (corresponding to polynomial g(x))

$$= 1 + x + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11}$$

then the desired code configuration is shown in Fig. 3.4. This scheme has been implemented.

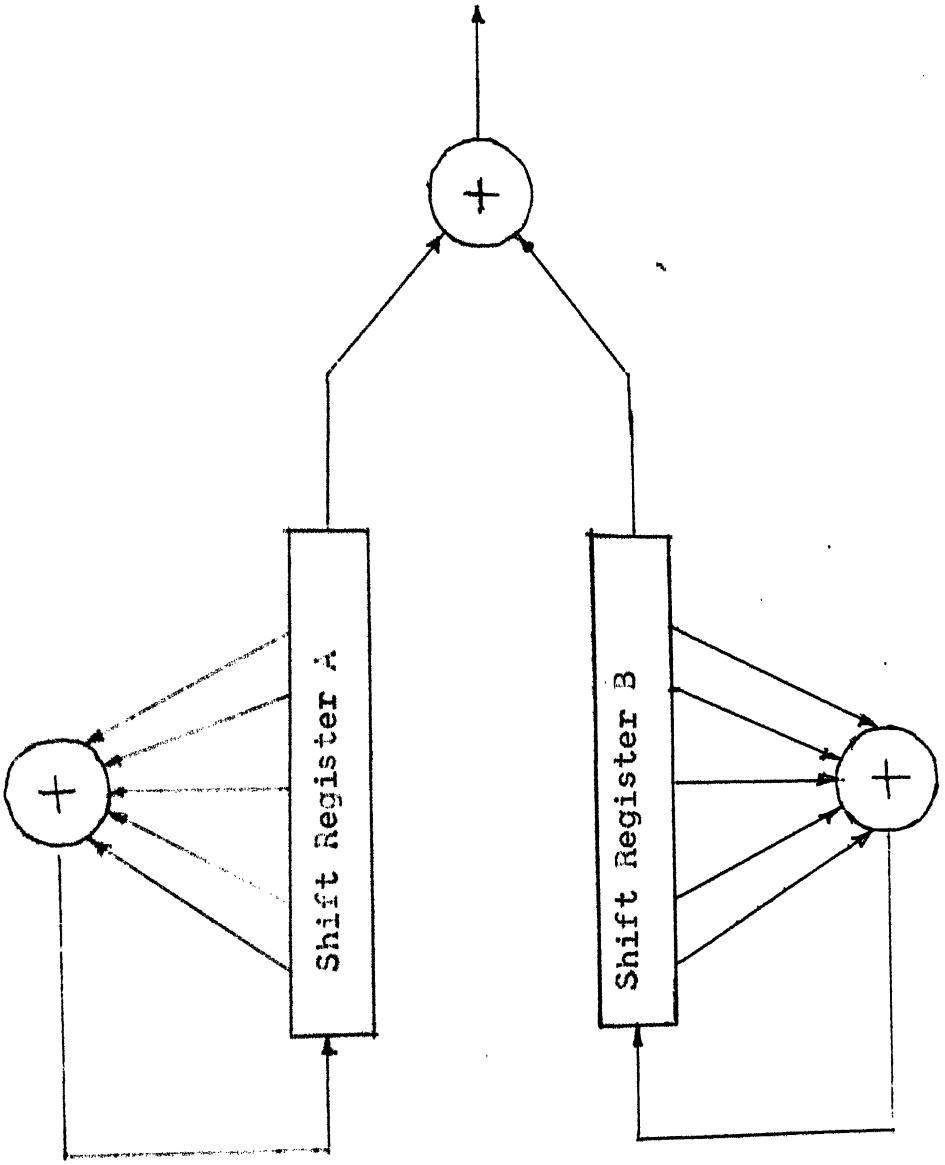


Fig. 3.3 Gold code Generator configuration.

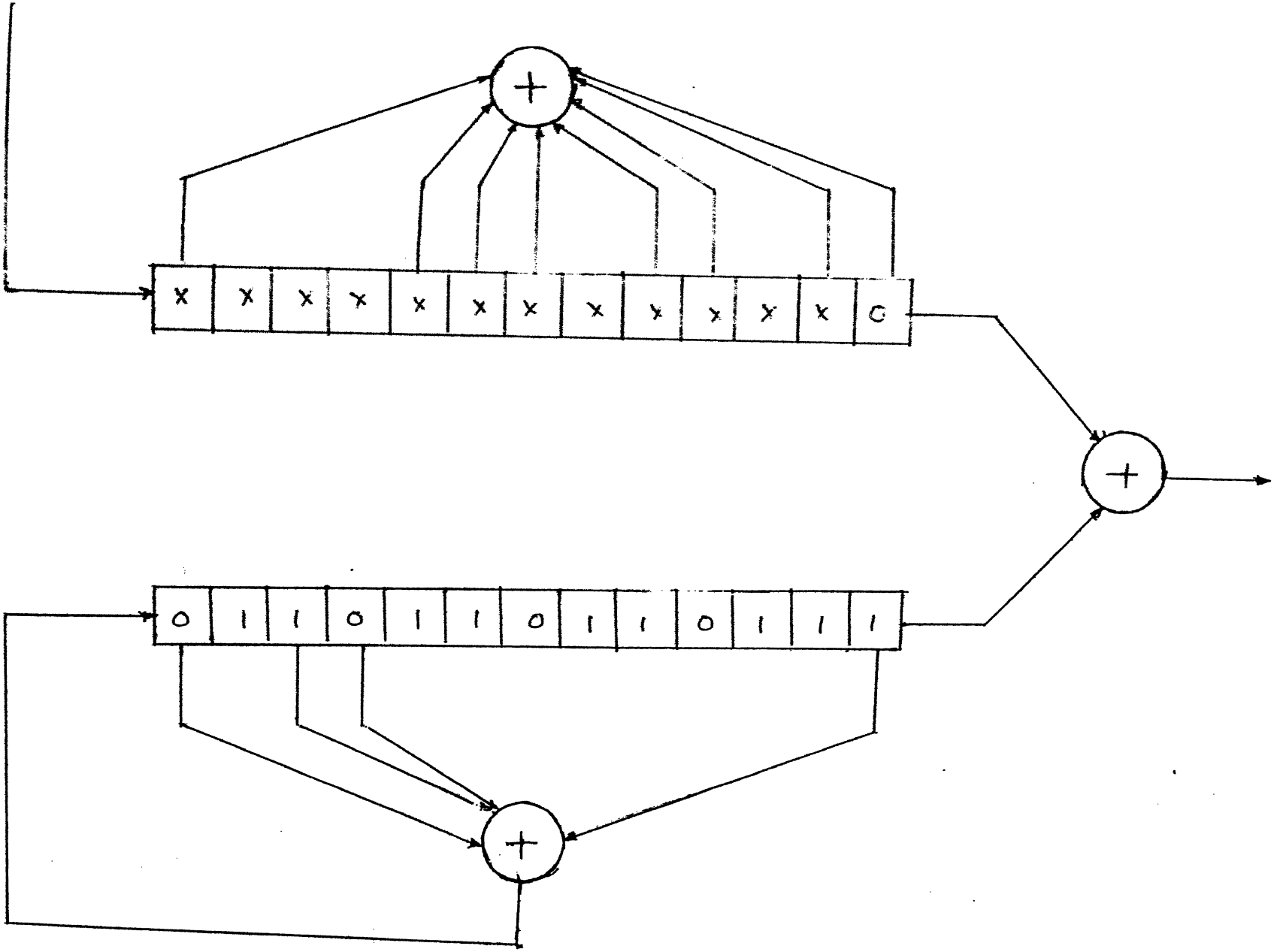


Fig. 3.4 Balanced Gold code of length $2^{13}-1$.

3.3 DELAY LOCK LOOP

The delay lock loop is a nonlinear feedback system which employs a form of cross-correlation in the feedback loop. Loop tracks the time varying phase of the receiver spreading waveform $p(t - T_d)$. The function $\hat{T}_d(t)$ denotes the receiver estimate of $T_d(t)$. T_d and $\hat{T}_d$ are always function of time. The received signal consists of the spreading waveform $\sqrt{P}\ p(t - T_d)$ with power P and additive white Gaussian noise n(t) with two-sided power spectral density $N_o/2$ W/Hz.

$$S_r(t) = \sqrt{P}\ p(t - T_d) + n(t)$$

Fig. 3.5 is a conceptual block diagram of the tracking loop. It consists of a phase discriminator, a loop filter, a voltage controlled oscillator and code sequence generator.

The received signal is correlated with an early spreading waveform $C(t - \hat{T}_d + (\Delta/2)\ T_c)$ and a late spreading waveform $C(t - \hat{T}_d - (\Delta/2)T_c)$. Δ is the total normal time difference between the early and late discriminator channels.

Assuming operation of the delay lock discriminator as a static phase measuring device in a noiseless environment i.e. T_d and $\hat{T}_d$ are fixed. The discriminator output will contain a component which is a function of

$$\delta = \frac{T_d - \hat{T}_d}{T_c}$$

output of early correlator

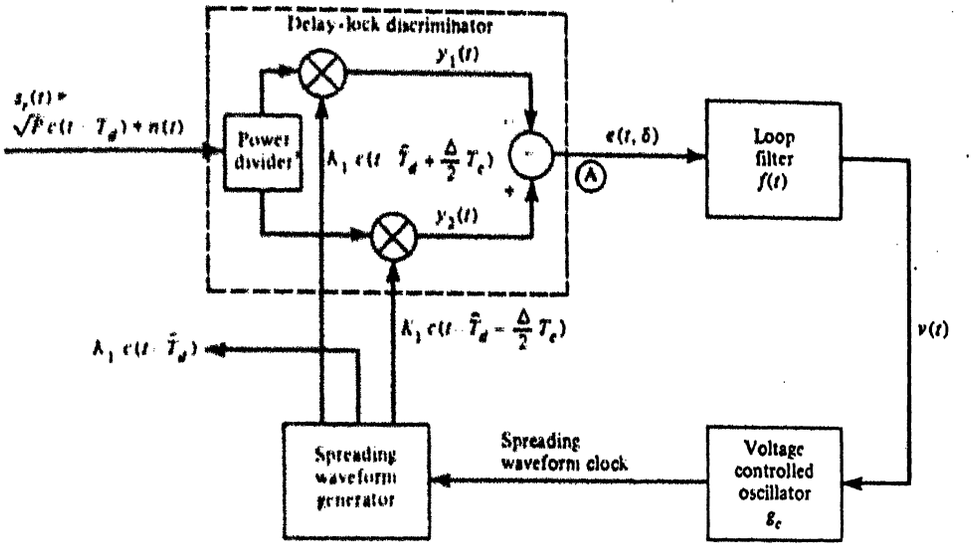


Fig. 3.5 Conceptual block diagram : baseband delay-lock tracking loop.

$$y_1(t, T_d, \hat{T}_d) = K_1 \sqrt{\frac{P}{2}}\, p(t - T_d)\, p(t - \hat{T}_d + \frac{\Delta}{2} T_c)$$

output of late correlator

$$y_2(t, T_d, \hat{T}_d) = \sqrt{\frac{P}{2}}\, p(t-T_d)\, p(t-\hat{T}_d - \frac{\Delta}{2} T_c)$$

In these expressions, K_1 is the multiplier gain and is dependent on the particular multiplier hardware implementation. The input signal has been divided by $\sqrt{2}$ to account for power division and noise has been ignored. Then error signal is

$$\epsilon(t, T_d, \hat{T}_d) = y_2(t, T_d, \hat{T}_d) - y_1(t, T_d, \hat{T}_d)$$

$$= K_1 \sqrt{\frac{P}{2}}\, p(t - T_d) \left[p(t - \hat{T}_d - \frac{\Delta}{2} T_c) - p(t - \hat{T}_d + \frac{\Delta}{2} T_c)\right]$$

The d.c. component of the error signal $\epsilon(t, T_d, \hat{T}_d)$ is used for code tracking. The time varying component, which is also a function of δ, is called self noise but it is filtered out in loop filter. Let d.c. component of $\epsilon(t, T_d, \hat{T}_d)$ is denoted by

$$K_1 \sqrt{\frac{P}{2}}\, D_\Delta(T_d, \hat{T}_d)$$

and $$K_1\sqrt{\frac{P}{2}}\, D_\Delta\, (T_d, \hat{T}_d) = \frac{1}{NT_c} \int_{-NT_c/2}^{NT_c/2} K_1 \sqrt{\frac{P}{2}}\, p(t-T_d) \left[p(t - \hat{T}_d - \frac{\Delta}{2} T_c) - p(t - \hat{T}_d + \frac{\Delta}{2} T_c)\right] dt$$

where NT_c is period of code p(t).

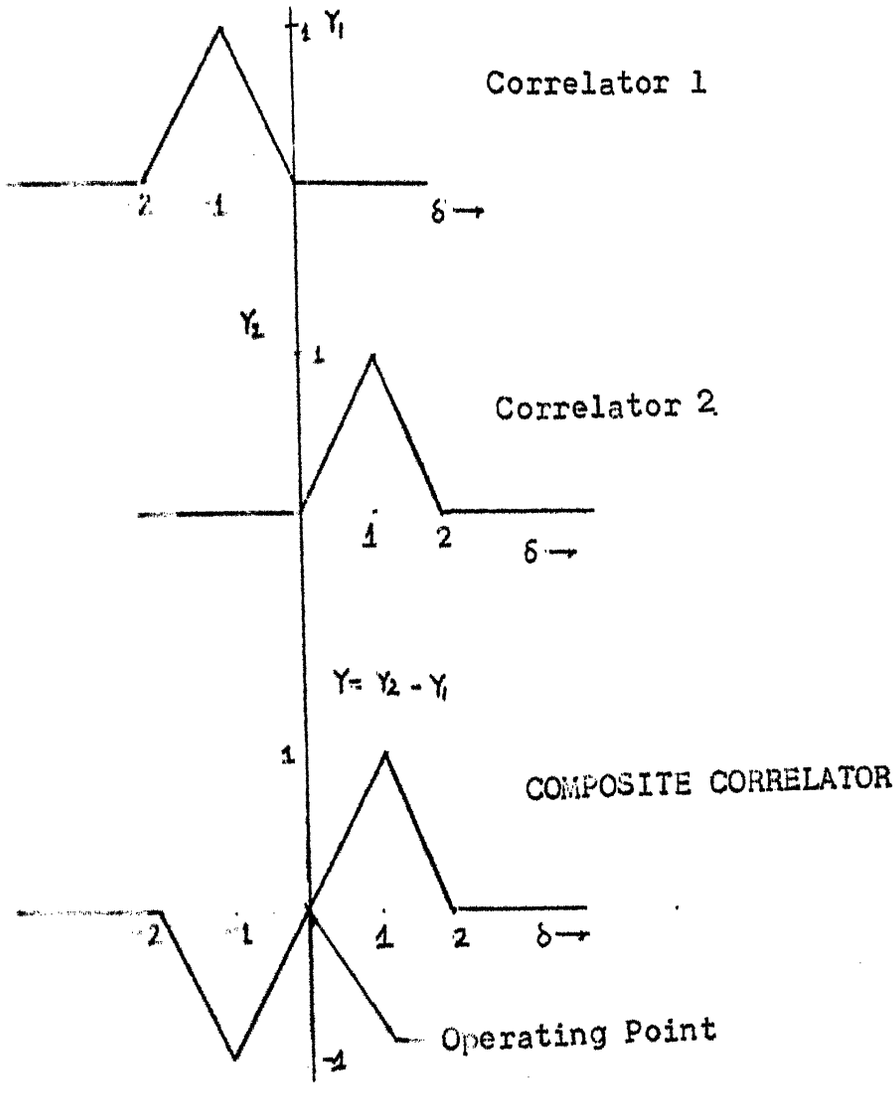


Fig. 3.6 Receiver's (DLL) Characteristic

So therefore, taking in account auto-correlation function of p(t)

$$D_\Delta\,(T_d,\ \hat{T}_d) = R_c\,(T_d - \hat{T}_d - \tfrac{\Delta}{2}\,T_c) - R_c\,(T_d - \hat{T}_d + \tfrac{\Delta}{2}\,T_c)$$

$$= R_c\left[(\delta - \tfrac{\Delta}{2})\,T_c\right] - R_c\left[(\delta + \tfrac{\Delta}{2})\,T_c\right]$$

$$\triangleq D_\Delta(\delta)$$

If we take $\Delta = 2$ bits then two locally genetaed sequences, namely early PN code and late PN code, are shifted by two bits. Then system characteristic is difference of two correlation functions. Response of two correlators and system characteristic is shown in Fig. 3.6. $D_\Delta(\delta)$ is linearly related to δ. Linear region is normal operating region for the tracking loop.

The summed correlated output is filtered and is used to control the receiver clock. The receiver code will track the incoming code at a point halfway between the maximum and minimum of the composite correlator output [14, Chapter 9, Page 420-428].

As we have discussed and developed appropiate structres for transmitter and receiver, in next chapter we will describe the hardware realizations of the developed system.

CHAPTER -4

SYSTEM REALIZATION

In the last chapter we discussed the system at the block diagram level and developed appropriate transmitter and receiver structures. Towards implementation of the scheme we discuss, here, the details of hardware realization.

We have implemented the transmitter circuit. A software program has also been developed to study the receiver structure. A brief description of transmitter hardware and receiver software is given below.

4.1 TRANSMITTER

4.1 (1) Brief Outlay

We implemented transmitter for information at data rate 64 kbps. This bit rate is selected in accordance of the bit rate of digitized voice signal. Since for voice digitization PCM modulation is generally used and rate of PCM modulated voice is 64 kbps, therefore bit rate of 64 kbps is selected for implementation of the transmitter. Operating frequency of the transmitter is choosen to be 4 MHz. PN code sequence is generated at 4 Mbps. At 4 MHz, we were able to use the existing TTL devices in store. Also, 4 MHz provides large enough processing gain (64). Higher frequency can be used if available components are fast enough.

Blocks of data at 4 Mbps can be used in the higher hierarchy of the digital transmission.

A basis block diagram of transmitter is shown in Fig. 4.1. Here analog voice signal is converted to digital data stream. Voice is digitized by PCM modulation.

Analog voice signal is passed through the PCM filter to limit the bandwidth of voice. Bandwidth limited voice signal is subsequently modulated by a PCM modulator. PCM modulator digitzes the analog voice signal. Output of PCM modulator constitutes the information. Data rate of signal is 64 kbps [since filter restricts the bandwidth upto 4 KHz and sampling rate is 2x4 = 8 kbps, and PCM modulator uses 8 bits for modulating one sampled value. Therefore, bit stream at the PCM modulator output is at 64 kbps (8 kbps x 8)].

In actual implementation of transmitter circuit we generated data at 64 kbps by a PN code generator of length 2^{6-1}. This pseudo random sequence replaced the digitized voice in Fig. 4.1. PN sequence is generated using maximal length linear feedback shift register of length 6. This data is added (Mod 2) to the PN code. Clock input to data generator is 64 kbps and is obtained by dividing 4 MHz clock.

We needed clock at 4 MHz frequency. For generating stable clock, a crystal is used. Since 4 MHz crystal was not available in store, a crystal of 8 MHz frequency was used. This clock of 8

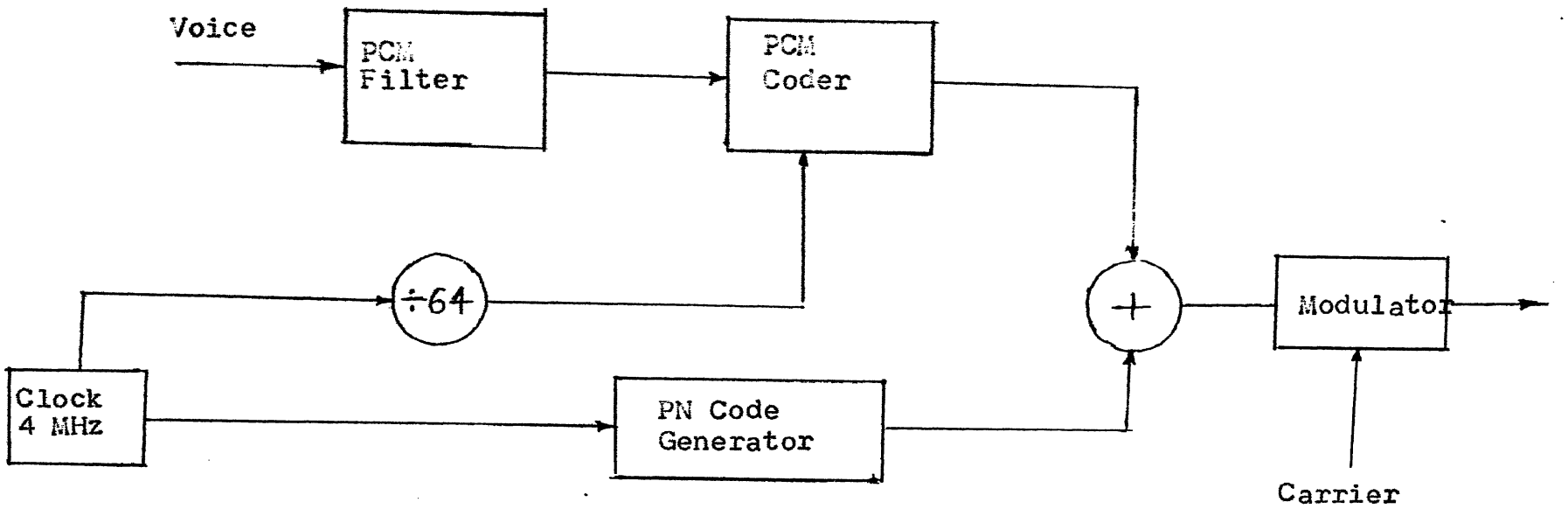


Fig. 4.1 System Transmitter.

MHz frequency is down converted for use in data generator (64 kbps) and in PN code generator (4 MHz).

4.1 (2) Hardware Description

In this section we will briefly described the details of transmitter. It will be discussed under following subsections :

(a) Clock generator

(b) Code generator

(c) Data transmitter

(d) Voice transmitter

Standard low power TTL devices are used for realization of various blocks, because highest operating frequency of the system (4 MHz) is well within the limiting frequency of standard TTL devices. For higher Frequency, fast TTL (74 F series) can be used.

(a) Clock Generator

Clock is generated using 8 MHz crystal. Crystal is used with inverter (74LS04) and other passive component. Circuit diagram of clock generator is given in Fig. 4.2. 74LS04 is a hex inverter. We have used two of its inverters. Crystal is connected between Pin 1 and 6 of 74LS04. Clock is generated at pin 6 of 74LS04. This is followed by a Schmitt trigger (74LS14) to smoothen out the ripples. A sketch of generated clock is shown in Fig. 4.2(b). Rise time is 18 nsec. and fall time is 25 nsec. Ripple of .4 V is observed in the clock.

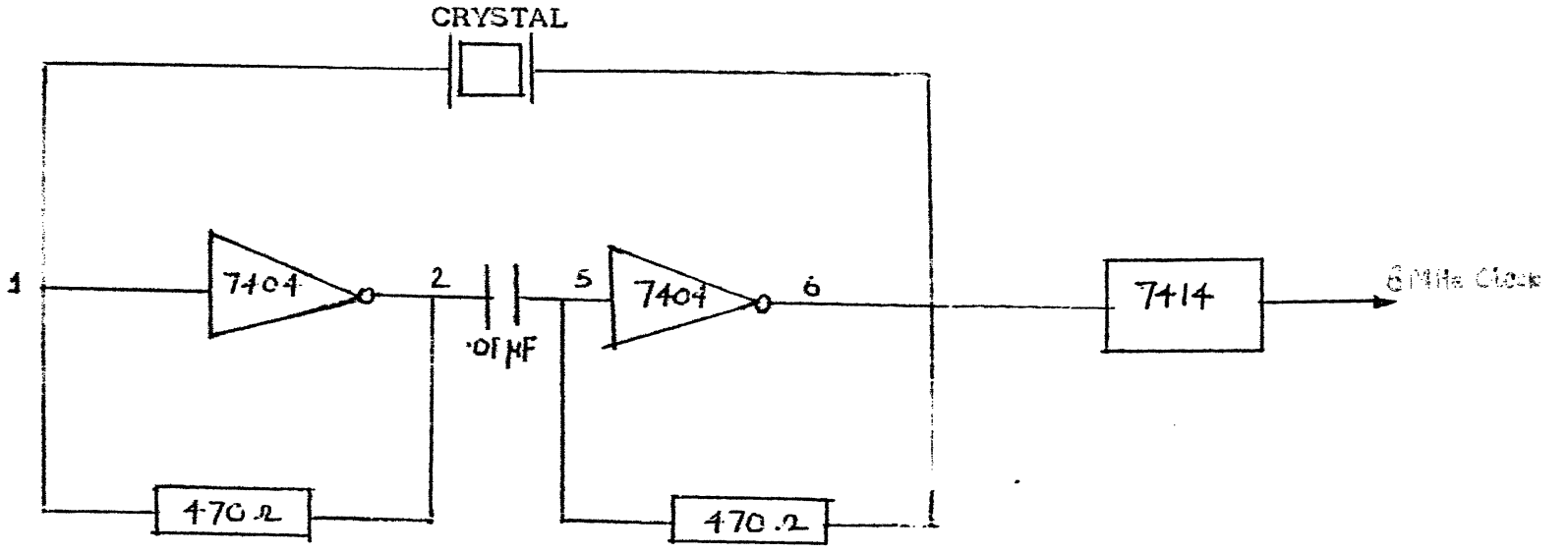


Fig. 4.2(a) Circuit Diagram of Clock Generator.

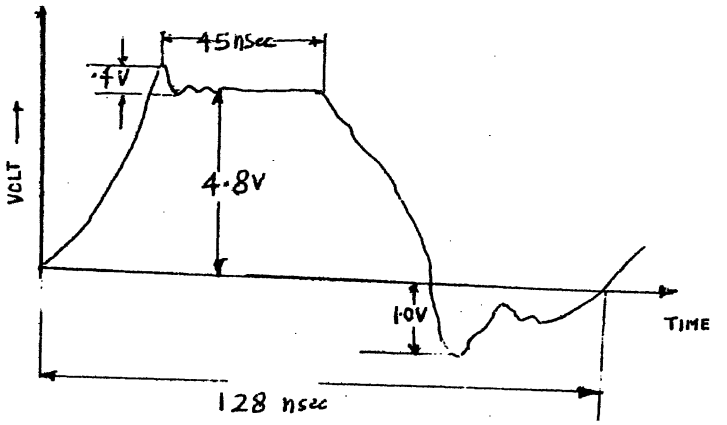


Fig. 4.2 (b) Clock waveform.

This clock is used as the base clock for the entire circuit.

(b) Code Generator

Structure of generator for Gold codes has been discussed in section 3.2. In Fig. 3.4, configuration for code generator is shown. It has two shift registers (1 and 2) of length 13. Initial conditions were also specified in Chapter 3. Shift register 1 and 2 are implemented using 74LS195 (4 bit parallel access shift register). Four 74LS195 are cascaded for each of the two registers. 74LS86's (EX OR) are used for EX OR operation (Mod 2 addition). It is needed to provide feedback connection. We can apply initial conditions to shift register by using parallel loading in 74LS195. Pin number 9 of 74LS195 is provided for this purpose. If signal at this pin 9 is low, parallel loading is accomplished and if signal is high then shifting is accomplished. Therefore pin number 9 has to kept low and initial conditions are applied at input pins (4, 5, 6, 7). As we apply clock pulse at pin number 10, these initial conditions are transferred to output of shift register and at this stage associated circuitry raises level of signal at pin 9 to high, thus enabling serial shifting. Pin number 2 and 3 are tied together and this is used as serial input. Associated circuit OR the outputs (thirteen) of register 2. If there is even single 1 in initial condition, then this circuit raises level of signal at pin 9 to high and initial condition always has some 1. OR operation for 13 inputs is implemented by 74LS27 (NOR gate), 74LS04 (NOT gate) and 74LS00

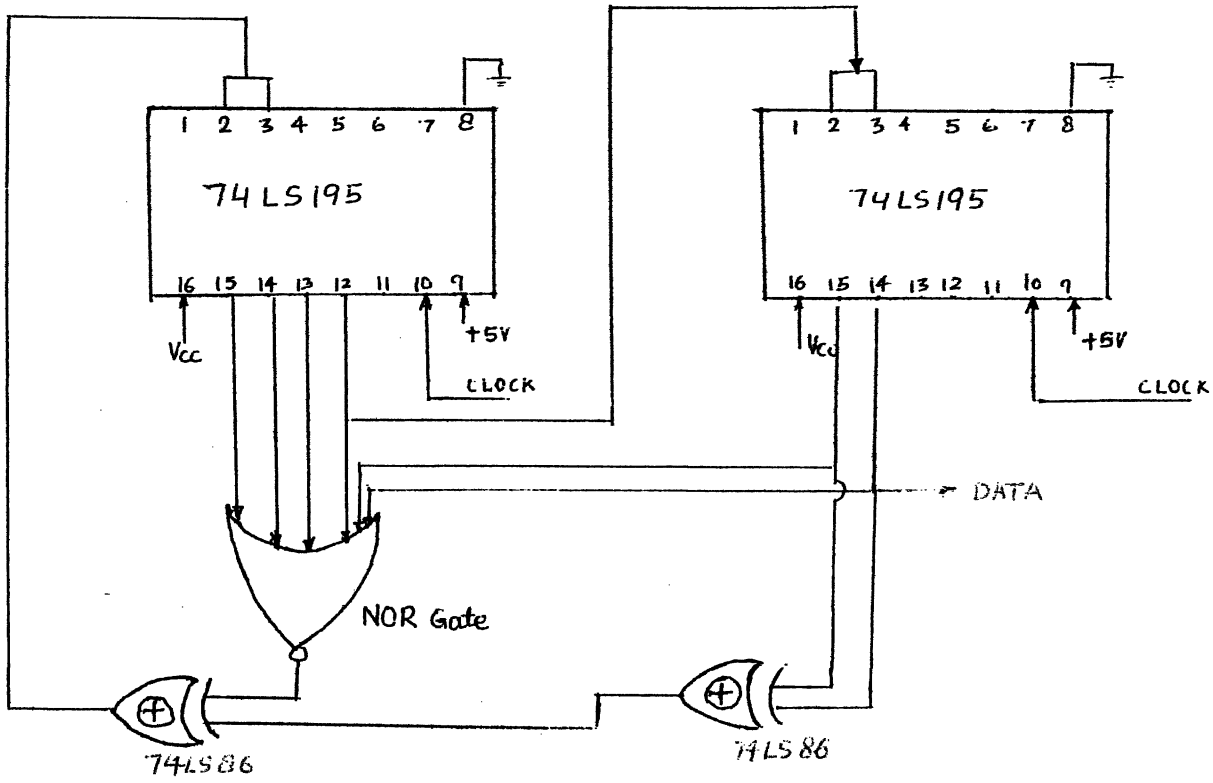


Fig. 4.3 DATA GENERATOR CIRCUIT.

all shift register and then EX ORing with the feedback. Now this is feeded back to shift register. If all zero occurs in shift register, then this provide a '1' to the shift register.

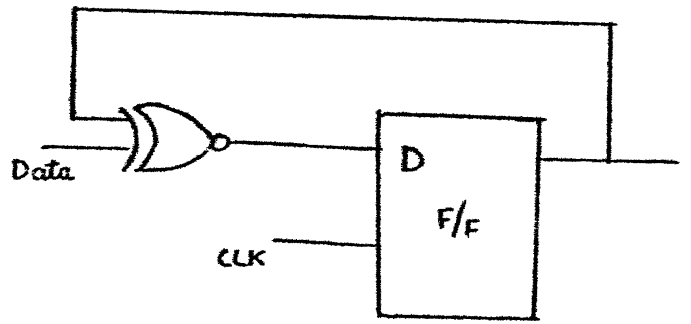
For data generator we need clock of 64 KHz. This is generated by dividing base clock of 8 MHz. For above purpose, synchronous 4 bit counters (74LS161) are used. Two 74LS161 are used in cascade such that it works as 6 bit synchronous counter. After the count of 63 (2^6-1) it resets the counter. Reset pulse is generated by 74LS30 (8 input NAND gate). Inputs to 74LS30 are parallel outputs of counter and when count is 63 then all inputs to 74LS30 are high (logical 1). Hence output of 74LS30 goes low and reset the count to zero. This reset pulse is applied at clear input of 74LS161. Counter's output are ORed to generate negative pulses of frequency 128 KHz (8 MHz/2^6). ORed output dips to 0 when counter's count is zero. So it gives negative pulse of duration 1/8 μsec. OR operation for 8 inputs is implemented by using 74LS27 and 74LS00. These pulses are used to generate 64 KHz clock by feeding them to clock input of edge triggered JK flip-flop (74LS112). This JK flip-flop is used in toggle mode i.e. J and K inputs are tied to 1. Output of 74LS112 is desired clock of frequency 64 KHz with duty cycle 50%. Clock is then passed through a line driver (74LS244) and is subsequently used to drive the data generator circuit. Circuit diagram of data transmitter is shown in appendix A.

Information data at 64 KHz is spreaded by PN Gold code at 4 Mbps in 74LS86 (EX OR) output is desired spreaded data and can be used to modulate a RF carrier.

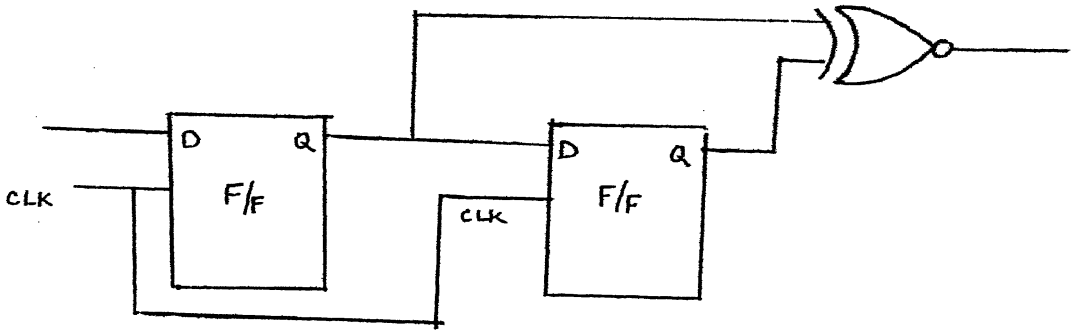
(d) Voice Transmitter

It is essentially the same as data transmitter with a difference of data generator. Here, instead of data generator, voice is quantized and PCM modulated. Intel 2912 is used as filter. 2912 requires clock frequency of 2 MHz and this is obtained by 74LS93 employed in code generator circuit. If we connect Q_o (pin 12) of 74LS93 to pin 1, then 74LS93 works as Mod 4, Mod 8, Mod 16 counter. Output is taken at pin 9 which gives 2 MHz clock. Filtered analog signal is passed to PCM coder. Circuit diagram is given in appendix A.

If baseband spreaded data is used to phase modulate RF carrier, then there is inherent phase ambigiuty in the demodulation circuit. This makes impossible to decide whether a current bit supposed to be a 1 or 0 without more information. This ambigiuty is due to reason that recovered carrier could have started in any of the phases. In BPSK modulator, this ambigiuty is resolved by encoding 1, 0 stream in such a way that a zero produce a change in level while a one will maintain the same level. Thus a phase change will result if the data bit is zero while a one will produce no phase change, which resolve the ambigiuty. Now this data is spreaded with PN code. Fig. 4.4



(a)



(b)

Fig. 4.4 Encoding (a) and Decoding (b) blocks for data ambiguity resolution

shows the encoding and decoding blocks for data ambiguity resolution.

Since we have used only baseband signal we do not need to implement the above discussed encoding and decoding blocks.

4.2 RECEIVER

Structure of the receiver was developed in section 3.3. In receiver we have to generate the replica of the PN code sequence used in transmitter. Receiver code sequence has to be in synchronism with the received signal. A scheme based on delay locked loop (DLL) was proposed in section 3.3. A schematic diagram of the proposed receiver is shown in Fig. 3.5. In this we used two correlators. Based on the output of these two correlators an error signal is generated which is used to drive the clock used in receiver characteristics of the receiver was shown in Fig. 3.6.

We developed a software program which simulates the operations of DLL. Program uses various functions which will be described latter on in this chapter. Flow chart of the program is given in Fig. 4.5. Fig. 4.5(a) shows the flow chart of the transmitter simulation. A PN code with the help of function MLC-GENERATE, is generated. Program accepts the data and adds it to PN code. Since information data rate and code data rate are different, information data is expanded by a factor of processing gain. Function EXPAND expands the data by a desired factor. Now data is called modulated data. To simulate the delay in channel,

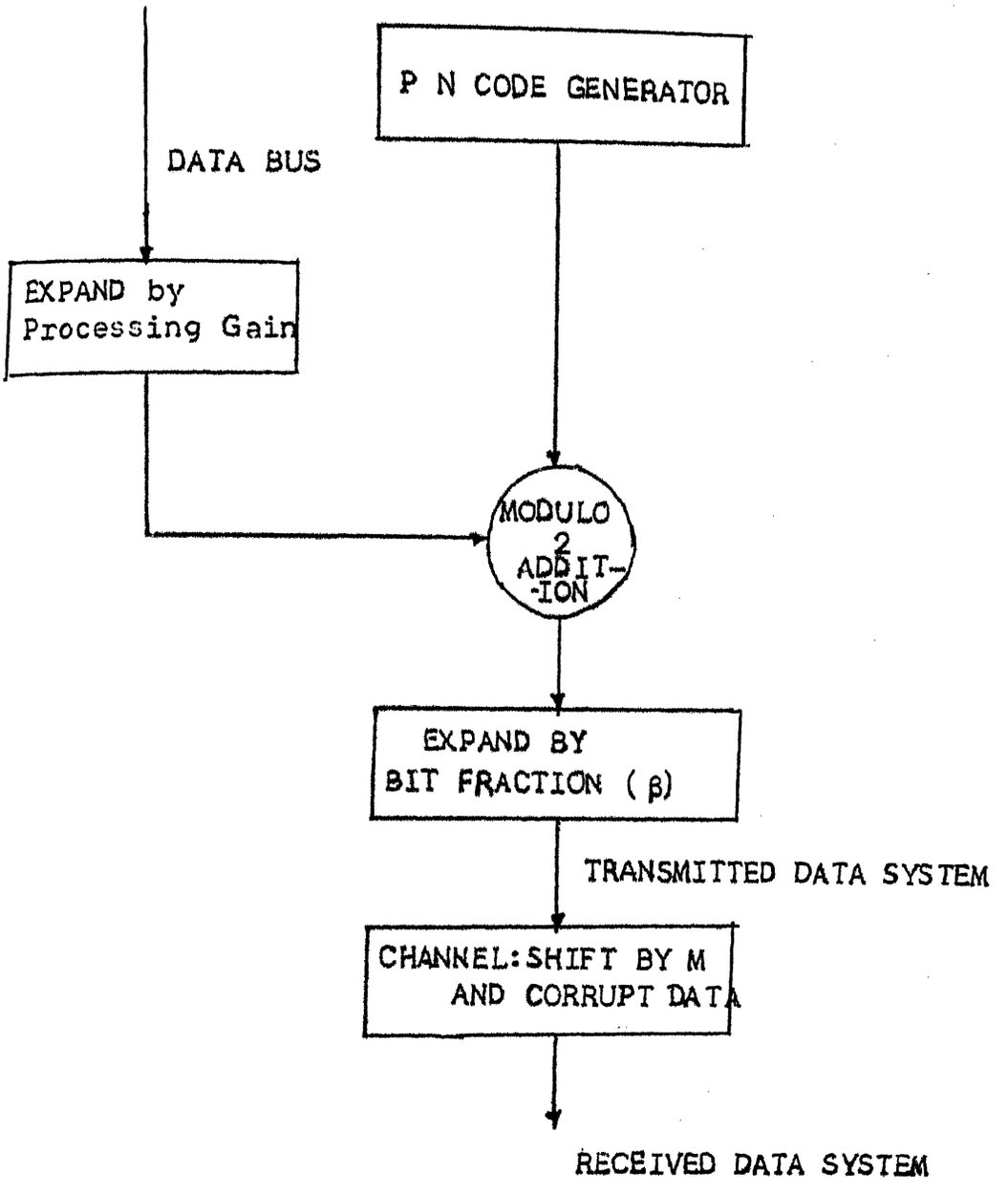


Fig. 4.5(a) TRANSMITTER FLOW CHART.

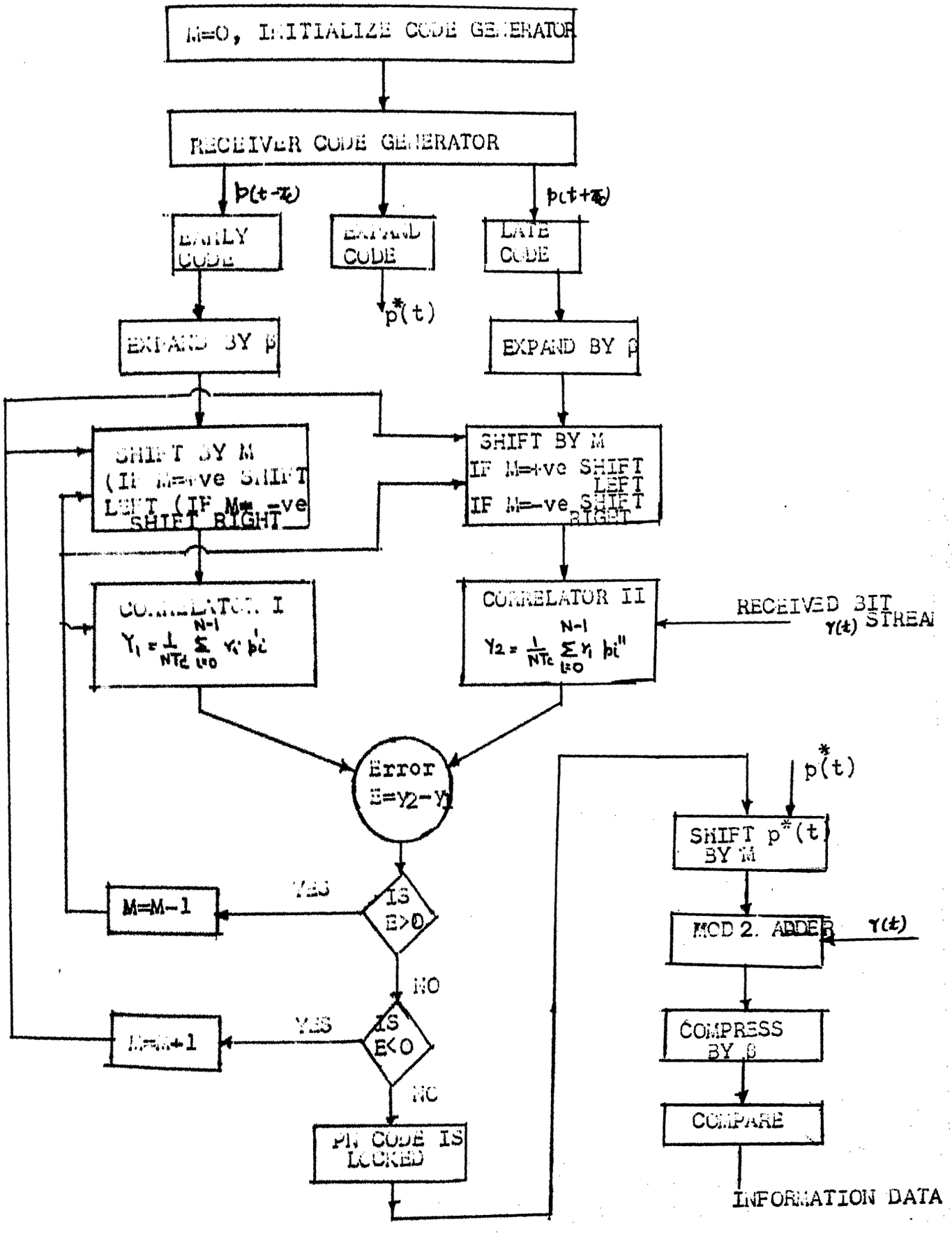


Fig. 4.5 (b) Receiver Flow Chart

each bit is expanded by a factor named bit fraction (β). Bit fraction (β) is programmed to be a variable parameter. Now one bit is represented by β bits and it is transmitted data. Transmitted data is now shifted by desired bits and desired delay is produced. One bit shift is equal to T_c/β delay. Function SHIFT shifts the data.

This delayed sequence is used as received signal in receiver. This can also be corrupted by another sequence. Receiver's flow chart is shown in Fig. 4.5(b). In receiver, we generate same PN code as used in transmitted. As described in section 3.3, two shifted versions (early and late codes) of PN sequence are generated. Early and late PN codes are expanded by β (as the transmitted sequence was expanded by same factor). Now expanded early and late codes are now shifted by M. M is a running parameter which depends on error signal. M is initialized to 0. These shifted codes are correlated with the received sequence. Function CORRELATE correlates the two sequences and gives d.c. value as output. Next error signal is calculated by subtracting two correlaters output. This error signal (E) is now checked for controlling the receiver clock. If E > 0, program decreases M by one, and returns back to shift the receiver code by M and again examine the error signal. If E < 0 then program increases M by one and shifts the receiver code by M. This process is repeated until error signal is nearly equal to zero. When E = 0, then receiver code is aligned to received signal. Now program shift p(t) by M and despread the received signal and

information data is obtained by compressing the demodulated data by β.

Various functions used in receiver are breifly described below :

MLC GENERATE

This function takes shift register size, initializing conditions and position of register at which clock is drawn, as input. This function generates clock as dictated by irreducible polynomial. Function also concatenate code sequence N times. Integer N is also input to function. N depends on number of data bits. Output of function is repeated code sequence.

FUNCTION EXPAND

This function accepts the sequence as input clock and gives out a sequence. Output sequence is expanded by a factor called expand-scale. Size of the input is also input to this function.

FUNCTION SHIFT

Output to this function are input sequence, sequence size and direction and magnitude of shift. If shift parameter is positive then function shifts the input sequence towards left and if shift parameter is negative the function shifts towards right. If shift parameter is zero then unshifted same sequence is returned.

FUNCTION CORRELATE

This function accepts two sequence and returns their correlated value as d.c. value. Clock size of input clocks is also given as input to function.

Program listing of above described software is appended in appendix.

CHAPTER - 5

CONCLUDING REMARKS

In this chapter, difficulties faced in implementing the scheme and efforts to overcome these difficulties are presented. We conclude by suggesting further work which could not be carried out due to paucity of time.

5.1 DIFFICULTIES FACED IN IMPLEMENTATION

In this work, we tried to exploit the multiple access capability of the spread spectrum technique. We implemented a transmitter circuit and generated a set of Gold codes, suitable for code division multiple access. Transmitter circuit was working satisfactorily. Frequency of operation was 4 MHz but it can be changed to higher value with use of faster devices. We also designed a transmitter for voice but it could not be implemented because of some problem in filter I.C. (Intel 2912). This I.C. worked properly but whenever we switched off the power supply, it developed some problem and did not work further. Probably it was very much sensitive to spikes present in power supply. We tried to reduce effect of spikes by using capacitors across the power supply. But we could not use this chip. Therefore we fabricated a data circuit to use in place of voice input. This data circuit was essentially a PRBS generator of length 63 ($2^6 - 1$).

Noise spikes were observed in the transitions of the generated PRBS from '0's to '1's and from '1's to '0's. To get clear PRBS, effective decoupling was used at every IC's V_{cc} terminal. This was done by using parallel combination of electolytic and ceramic capacitors. Both the capacitors should be used to provide smooth filtering over a wide range of frequencies. One of the capacitors must be of higher value than the other capacitor. Using this combination transition in PRBS were reduced.

Due to paucity of time we could not implement the hardware for receiver. But the functions of DLL receiver were studied by a software simulation. If the time delay in channel was within 2 bits then DLL was able to track the received signal. Therefore, acquisition circuit should be able to limit the delay error to two bit time. It was also concluded that when number of users were below the upper bound, demodulation was nearly error free.

5.2 SCOPE OF FURTHER IMPROVEMENT

System can be further improved by incorporating certain modifications. Firstly, since the receiver was not implemented, receiver can be implemented either by using the proposed scheme or by using some other scheme. Specific dedicated chips could also be used to optimise the circuit design. These special chip could be crystal controlled oscillators, SAW delay lines etc. These dedicated chips can be very useful in high frequency applications.

Error correction coding can also be used to improve the performance of system. As an indication of the improvement that can be obtained with appropriate error-correction techniques, a table is given below [1]

Table 5.1 : Comparison of coding techniques

Technique	Required channel P_o	Required E_b/N_o	Complexity (no. of ICs)
No coding	10^{-3}	11.7	-
r = 1/2, conv.	0.0266	10.2	10
Golay (24, 12)	0.0266	10.2	20
Block (48, 24)	0.029	10.0	60
r = 1/3, conv.	0.082	10.7	(Viterbi) 80
r = 1/2, conv.	0.064	9.4	(Seq.) 500

By using error corection codes, higher channel error probability can be tackled with reduced required E_b/N_o. Ofcourse complexity of the system increases quite significantly. Convolutional codes are extremely efficient in this regard, although the complexity of the decoders may become quite significant. Attempts should be made for implementation of systems with higher processing gain.

REFERENCES

[1] George R. Cooper and Clare D. McGillem, "Modern Communications and Spread Spectrum", McGraw-Hill International Edition 1986.

[2] R.A. Scholtz, "The Origins of Spread-Spectrum Communications", IEEE Transactions on Communication, Vol. Com-30, No. 5, May 1982.

[3] D.V. Sarwati and M.B. Pursley, "Cross Correlation Properties of Pseudo-random and Related Sequences", Proc. IEEE, Vol. 68, May 1980.

[4] Harrison E. Rowe, "Bounds on the Numbers of Signals with Restricted Cross Correlation", IEEE Transactions on Comm. Vol. Com-30, No. 5, 1982.

[5] Charles L. Weber et al., "Performance Consideration of Code Division Multiple-access Systems", IEEE Trans. on Vehicular Technology, Vol. VT-30, No. 1, Feb. 1981.

[6] Urs Grob et al., "Micocellular Direct-sequence Spread Spectrum Radio System using N. Path RAKE Receiver", IEEE Journal on Selected areas in communications, Vol. 8, No. 5, June 1990.

[7] E. Geraniotis et al., Broadcast Capability of Direct-sequence and Hybrid Spread Spectrum", IEEE Journal on Selected areas in communications, Vol. 8, No. 4, May 1990.

[8] Donald C. Kemdirim and Jim S. Wight, "DS SSMA with some IC Realizations", IEEE Journal on Selected areas in communications, Vol. 8, No. 5, June 1990.

[9] Maj V.J.S. Lidder, "A High Data Rate Baseband Spread Spectrum Delay Lock Receiver", M.Tech. Project 1988.

[10] Sq. Leader S.K. Bhartari, "A High Data Rate Baseband Spread Spectrum Transmitter", M.Tech. Project 1988.

[11] Raymond L. Pickholtz et al., "Theory of Spread Spectrum Communications - A Tutorial", IEEE Transaction on Comm. Vol. Com-30, No. 5, May 1982.

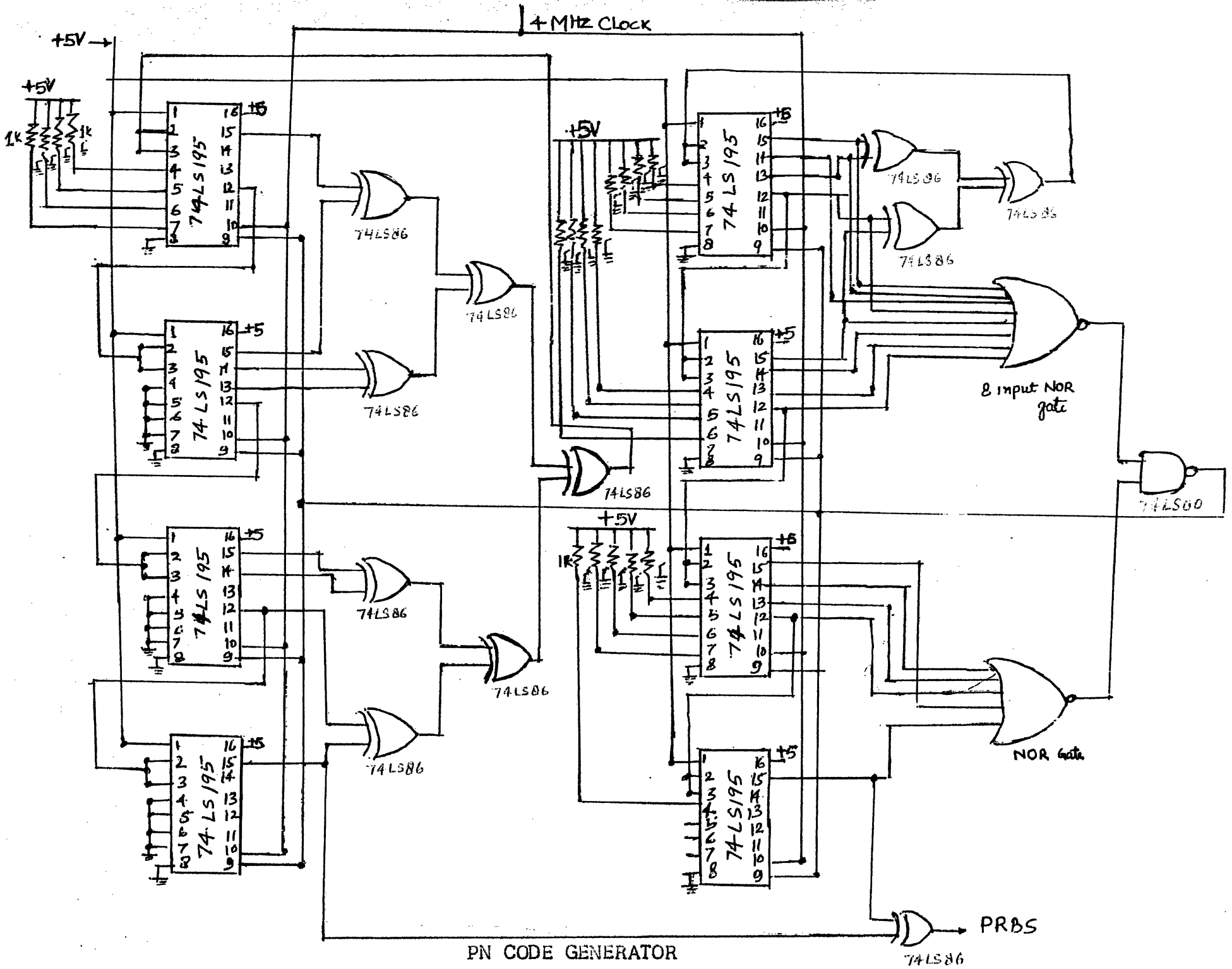
[12] Robert M. Gagliardi, "Satellite Communication", CBS Publishers & Distributors.

[13] Jack K. Holmes, "Coherent Spread Spectrum Systems", John Wiley and Sons, Inc. 1982.

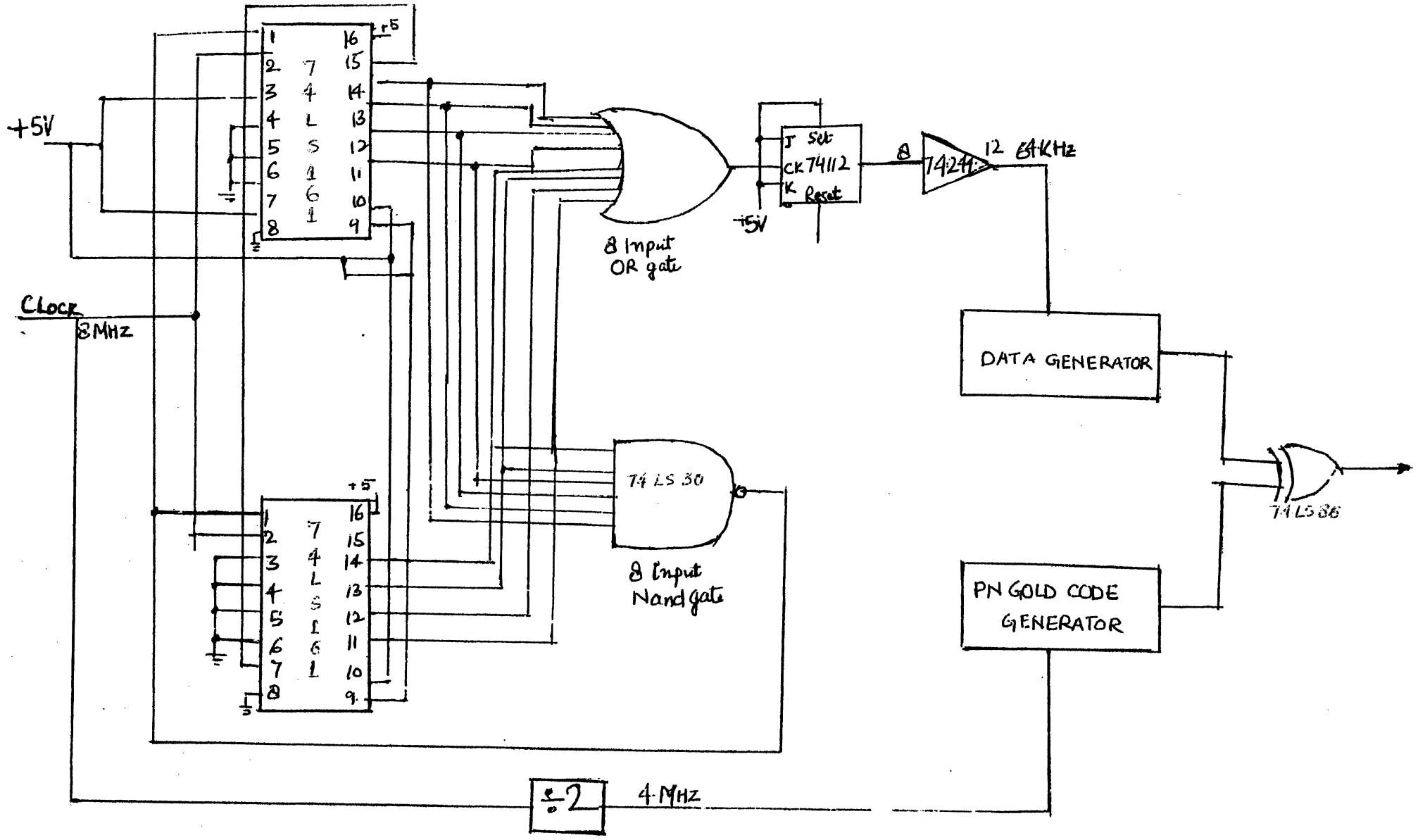
[14] W.W. Peterson and E.J. Weldon Jr., "Error Correcting Codes", 2nd edition M.I.T. Press 1972.

[15] Rodger E. Ziemer and Roger L. Peterson, "Digital Communications and Spread Spectrum Systems", Macmillan Publishing Company 1985.

[16] TTL Logic Data Manual.



PN CODE GENERATOR



DATA TRANSMITTER CIRCUIT

6 " MAIN PROGRAM "

```
#include <stdio.h>
#define counter_size 6
#define bit_fraction 10
#define data_size 3
#define LIMIT 10
int initialize[10];
int data[100],demod_data[10000];
int code[1000],expand_data[1000],modulate_signal[1000];
int clock_rx0[1000],clock_rx1[1000],clock_rx2[1000];
int expand_modulate[10000],clock_rx0_expand[10000],clock_rx2_expand[10000],clock_rx1_expand[10000];
int tmp_clock0[10000],tmp_clock1[10000],tmp_clock2[10000];
int rx_signal[10000];
int dc_value;

main()
        {int i,j,k;
        int dc_value_1,dc_value_2;
        int seq_length,
        int shift_tx,shift_rx;

        int lock=-1;/*=0 for locking of rx else = -1 */

        shift_rx=0;


        printf("i/p channel delay in terms of bit_fraction  ");
        scanf("%d",&shift_tx);


        initialize[0]=1;
        initialize[1]=0;
        initialize[2]=0;
        initialize[3]=0;
        initialize[4]=0;
        initialize[5]=0;
        data[0]=0;
        data[1]=1;
        data[2]=0;
        data[3]=0;
        data[4]=1;
        data[5]=0;
        data[6]=0;

        seq_length=power(2,counter_size)-1;

        mlc_generate(&code,1,initialize,counter_size,data_size+1);
        printf("mlc sequence");
        for(i=seq_length*(data_size+1)-1;i>=0;i--)
                printf("%d",code[i]);
        expand(&data,&expand_data,seq_length+1,data_size);
        printf("\n data =  ");
        for(i=data_size-1;i>=0;i--)
                printf("%8d",data[i]);
        printf("\nexpanded data = ");
        for(i=data_size*(seq_length+1)-1;i>=0;i--)
                printf("%d",expand_data[i]);
        for(k=0;k(=(seq_length+1)*data_size-1;k++)
          modulate_signal[k]=expand_data[k]^code[k];
        printf("\nmodulated sig = ");
        for(i=data_size*(seq_length+1)-1;i>=0;i--)
                printf("%d",modulate_signal[i]);
        expand(&modulate_signal,&expand_modulate,bit_fraction,(seq_length+1)*data_size);
        printf("\ntransmit sign = ");
        for(i=(bit_fraction*(seq_length+1)*data_size-1);i>=0;i--)
                printf("%d",expand_modulate[i]);
        shift(&expand_modulate,&rx_signal,bit_fraction*data_size*(seq_length+1),shift_tx);
```

```
        printf("\nreceived sign = ");
        for(i=bit_fraction*(seq_length+1)*data_size-1;i>=0;i--)
                printf("%d",rx_signal[i]);
        mlc_generate(&clock_rx0,0,initialize,counter_size,data_size+1);
        printf("\nclock_rx0 = ");
        for(i=seq_length*(data_size+1)-1;i>=0;i--)
                printf("%4d",clock_rx0[i]);
        mlc_generate(&clock_rx2,2,initialize,counter_size,data_size+1);
        printf("\nclock_rx2 = ");
        for(i=seq_length*(data_size+1)-1;i>=0;i--)
                printf("%4d",clock_rx2[i]);
        expand(&clock_rx0,clock_rx0_expand,bit_fraction,(data_size+1)*seq_length);
        printf("\nexpand_c_rx0 = ");
        for(i=bit_fraction*(data_size+1)*seq_length-1;i>=0;i--)
                printf("%d",clock_rx0_expand[i]);

        expand(&clock_rx2,clock_rx2_expand,bit_fraction,(data_size+1)*seq_length);
        printf("\nexpand_c_rx2 = ");
        for(i=(bit_fraction*(data_size+1)*seq_length-1);i>=0;i--)
                printf("%d",clock_rx2_expand[i]);


while(lock!=0)
{


        shift(&clock_rx0_expand,&tmp_clock0,bit_fraction*seq_length*(data_size +1 ),shift_rx);
        printf("\n shifted and expand_c_rx2 = ");
        for(i=(bit_fraction*(data_size+1)*seq_length-1);i>=0;i--)
                printf("%d", tmp_clock0[i]);

        shift(&clock_rx2_expand,&tmp_clock2,bit_fraction*seq_length*(data_size+1),shift_rx);
        printf("\ncorrelated clock1 = ");
        correlate(&tmp_clock0,&rx_signal,seq_length*bit_fraction,&dc_value_1);
        printf("\ncorrelated clock2 = ");
        correlate(&tmp_clock2,&rx_signal,seq_length*bit_fraction,&dc_value_2);
        printf("\ndc_value_1 = %3d    dc_value_2 = %3d ",dc_value_1,dc_value_2);
        dc_value=dc_value_1 - dc_value_2;

        if(dc_value>LIMIT)
                shift_rx++;
        else if(dc_value<-LIMIT)
                shift_rx--;
        else
                {
                lock=0;
                }

        printf("\n shift = %2d   receiver shift= %2d   dc_value= %2d",shift_tx,shift_rx,dc_value);


}
        printf(" \n\n\n\n\n\n\n\t\tL O C K E D !      Receiver shift = %3d \n\n ", shift_rx);
        mlc_generate(&clock_rx1,1,initialize,counter_size,data_size+1);
        printf("\nclock_rx1 = ");
        for(i=seq_length*(data_size+1)-1;i>=0;i--)
                printf("%4d",clock_rx2[i]);
        expand(&clock_rx1,clock_rx1_expand,bit_fraction,(data_size+1)*seq_length);
        shift(&clock_rx1_expand,&tmp_clock1,bit_fraction*seq_length*(data_size+1),shift_rx);
        printf("\n\n\n D E M O D U L A T E D   D A T A   \n\n");
        for(i=0,i<=bit_fraction*(seq_length+1)*data_size-1;i++)
        {
          demod_data[i]=tmp_clock1[i] ^ rx_signal[i];
          printf("%d",demod_data[i]);
```

```
                    * FUNCTIONS USED BY MAIN PROGRAM *
shift(in_signal,op_signal,signal_size,direction)
int direction;/* =-1 -->shift right; =1 -->shift left*/
int signal_size;/* size of in_signal sequence */
int *in_signal, *op_signal;
{
        int i,j;
        if(direction==0)
        for(j=0;j<=signal_size-1;j++)
                *(op_signal + j)= *(in_signal +j);

        if(direction)=1)
        {
        for(j=0;j<=direction-1;j++)
                *(op_signal+j)=0;
        for(i=0;i<=signal_size-direction-1;i++)
                *(op_signal+i+direction)=*(in_signal+i);
        }
        if(direction<=-1)
                {
                direction=-direction;
                for(j=0;j<=direction-1;j++)
                        *(op_signal+signal_size - j )= 0;
                for(i=0;i<=signal_size-direction-1;i++)
                *(op_signal+ i)=*(in_signal +direction+ i);
                }

}

correlate(clock1,clock2,clock_size,dc_value)
int *clock1,*clock2;
int clock_size,*dc_value;
{
        int i,j;
        int corr_clock[1000];
        *dc_value=0;
        for(i=0;i<=clock_size-1;i++)
                {
                corr_clock[i]=*(clock1 + i) & *(clock2 + i);
                printf("%d",*(corr_clock+i));
                *dc_value=*dc_value + corr_clock[i];
                }

}

expand(in_clock,op_clock,expand_scale,size_input)
int *in_clock,*op_clock;
int expand_scale, size_input;
{
        int i,j;

        for(j=0;j<=size_input-1; j++)
        {
          for(i=0;i<=expand_scale - 1; i++)
          *(op_clock + j*expand_scale + i) = *(in_clock + j);
          }
}

mlc_generate(clock,bit,initialize,counter_size,multiply)
int *clock;
int bit;/* gives the tap for the o/p sequence */
int multiply, /* factor for which multiplication of ml - sequnce*/
int initialize[10];
{
        int i,j,temp,
        int b[10];
        int clock_size ;
```

```
        clock_size=power(2,counter_size) -1;
        for(i=0;i(=counter_size-1 ;i++)
                b[i]=initialize[i];
        for(i=0;i(=clock_size-1; i++)
        {
          *(clock+i) =b[bit];
          temp=b[0]^b[1];
          b[0]=b[1];
          b[1]=b[2];
          b[2]=b[3];
          b[3]=b[4];
          b[4]=b[5];
          b[5]=temp;
        }
        for(j=1;j(=multiply;j++)
        {
          for(i=0;i(=clock_size-1;i++)
                *(clock+clock_size*(j-1) +i) = *(clock+i);

        }
}

int power(x,n)/*calculates x^^n where x is integer and n is positive integer*/
int x,n;
{
        int i,p;
        p=1;
        for(i=n;i)=1;i--)
                p=p*x;
        return(p);

}
```

```
i p channel delay in terms of bit_fraction  mic sequence=
1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1
1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1
0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0
1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1
0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0
1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0
1 1 0 0 0 0 0 1 0 0 0 0 0
 data =
     0     1     0
expanded data =
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
mudulated sig =
0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0
1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 1
1 0 0 1 0 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 1 1 0 1 0 1 1 1 0 0 1
1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0
1 1 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0
transmit sign =
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
received sign =
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1
1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
clock_rx0 =
 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1
```

```
1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0
1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1
0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0
0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1
1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1
1 0 0 0 0 1 0 0 0 0 0 1
clock_rx2 =
0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0
1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1
1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0
0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1
1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1
0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0
0 1 1 0 0 0 0 1 0 0 0 0
expand_c_rx0 =
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1
expand_c_rx2 =
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
```

```
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
 shifted and expand_clrx2 =
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
correlated clock1 =
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
1 1 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 0
correlated clockx2 =
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 0
dc_value_1 =  64    dc_value_2 =  96
 shift = -2   receiver shift= -1   dc_value= -32
 shifted and expand_c_rx2 =
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1
1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1
0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
```

```
0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
correlated clock1 =
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0
correlated clock2 =
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 0 0 0
dc_value_1 =  64    dc_value_2 =  80
 shift = -2   receiver shift= -2   dc_value= -16
 shifted and expand_c_rx2 =
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1
correlated clock1 =
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1
correlated clock2 =
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
```

```
0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0
dc_value_1 =  64    dc_value_2 =  64
 shift = -2   receiver shift= -2   dc_value=  0
```

```
              L O C K E D !      Receiver shift =  -2
```

```
clock_rx1 =
0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0
1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1
1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0
0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1
1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1
0 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0
0 1 1 0 0 0 0 1 0 0 0 0
```

```
 D E M O D U L A T E D   D A T A
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 
```